PATENT
Attorney Docket No.: 16869S-089800US
Client Ref. No.: W1094-01

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

KOJI SONODA et al.

Application No.: 10/637,216

Filed: August 8, 2003

For: COMPUTER SYSTEM

Customer No.: 20350

Examiner: Unassigned

Technology Center/Art Unit: 2131

Confirmation No.: 3324

PETITION TO MAKE SPECIAL FOR NEW APPLICATION UNDER M.P.E.P. § 708.02, VIII & 37 C.F.R. § 1.102(d)

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This is a petition to make special the above-identified application under MPEP § 708.02, VIII & 37 C.F.R. § 1.102(d). The application has not received any examination by an Examiner.

(a) The Commissioner is authorized to charge the petition fee of $130 under 37 C.F.R. § 1.17(i) and any other fees associated with this paper to Deposit Account 20-1430.

08/10/2004 WABDELR1 00000075 201430 10637216
01 FC:1460 130.00 DA

(b) All the claims are believed to be directed to a single invention. If the Office determines that all the claims presented are not obviously directed to a single invention, then Applicants will make an election without traverse as a prerequisite to the grant of special status.

(c) Pre-examination searches were made of U.S. issued patents, including a classification search and a key word search. The classification search was conducted on or around June 29, 2004 covering Classes 707 (subclass 200), 709 (subclass 220), and 711 (subclasses 100, 112, 114, and 156), by a professional search firm, Lacasse & Associates, LLC. The key word search was performed on the USPTO full-text database including published U.S. patent applications. The inventors further provided a reference considered most closely related to the subject matter of the present application (see references #6 below), which was cited in the Information Disclosure Statement filed with the application on August 8, 2003..

(d) The following references, copies of which are attached herewith, are deemed most closely related to the subject matter encompassed by the claims:

(1) U.S. Patent Publication No. 2001/0029507 A1;
(2) U.S. Patent Publication No. 2003/0046369 A1;
(3) U.S. Patent Publication No. 2003/0225972 A1;
(4) U.S. Patent Publication No. 2004/0015520 A1;
(5) Japanese Patent Publication No. 2003-015931; and
(6) John Kubiatowicz et al., "OceanStore: An Architecture for Global-Scale Persistent Storage," Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), November 2000, pp. 190-201.

(e) Set forth below is a detailed discussion of references which points out with particularity how the claimed subject matter is distinguishable over the references.

A. Claimed Embodiments of the Present Invention

The claimed embodiments relate to technology for a dispersed type of computer system and, more particularly, to a computer system in which there is flexibility to add or delete file attributes.

Independent claim 1 recites a computer system comprising a first storage device storing file management information, a first computer connected to the first storage device, a second computer connected to the first computer via a network, and a second storage device storing file data managed by the file management information, connected to the second computer.

Independent claim 14 recites a method of managing data. The method comprises storing file management information in a first storage device; connecting a first computer to the first storage device; connecting a second computer to the first computer via a network; storing file data in a second storage device; connecting the second storage device to the second computer; and managing the file data stored in the second storage device using the file management information stored in the first storage device.

One benefit that may be derived is the flexibility to add file attributes for each file and efficient management of access rights information for multiple users. Where a file is stored using a plurality of storage devices managed by servers dispersed in a plurality of sites, embodiments of the invention provide a computer system in which accounting information can be managed in respect of each site and each server.

B. Discussion of the References

None of the following references disclose or suggest a first storage device storing file management information and connected to a first computer; and a second storage device connected to a second computer which is connected to the first computer and storing file data managed by the file management information.

1. U.S. Patent Publication No. 2001/0029507 A1

This reference discloses a database-file link system and method therefor. A file server 20 and a database DB server 30 are connected through a communication network 90. A content information file 244 retains attribute information of the DB managed contents. During database registration, an OS file accessing control information 245 (for DB managed contents 243) is updated by means of the File Management System (FMS) 242. The OS file accessing control information 245, DB managed contents 243, File Management System 242, OS 241, and contents information file 244 all reside in the file server 20. See Figs. 1-3; and paragraphs [0039], [0040], [0043], [0074], [0078], [0080], [0084], [0090], [0091], and [0093].

2. U.S. Patent Publication No. 2003/0046369 A1

This reference relates to method and apparatus for initializing a new node in a network. A storage system (content repository) 1530 includes content provider's account information, assigned content management server, reserved storage, number of media files, and media file's attributes. A file metadata database within the storage system 1530 holds file metadata related to block files (block size, attributes, etc.). See Fig. 15; and paragraphs [0208], [0209], [0211], and [0215].

3. U.S. Patent Publication No. 2003/0225972 A1

This reference discloses a storage system in which a file attribute control unit 1331 and storage unit 143 executing a processing are linked together in response to a request from client computer(s) 11a-11b. The host computer 13 executes a file attribute control program 1331 to add a particular attribute to a file. The storage unit 143 operates in response to the added attribute. See Figs. 1 and 4; and paragraphs [0033]-[0037], [0040], [0041], and [0049]-[0051].

4. U.S. Patent Publication No. 2004/0015520 A1

This reference discloses database managing method and system having data backup function and associated programs. A database management program 10 has a data attribute changing module 140 for registering a data attribute or changing a data attribute

stored in a data attribute table 35. See Figs. 1 and 4; and paragraphs [0015], [0018], and [0019].

5. Japanese Patent Publication No. 2003-015931

This reference relates to information processing system and storage area providing method to provide a data storage system in consideration of the attributes (performance and cost). A plurality of storage devices have different attributes; and an attribute holding means is provided for holding information representing attribute of each storage device.

6. John Kubiatowicz et al., "OceanStore: An Architecture for Global-Scale Persistent Storage," Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), November 2000, pp. 190-201.

This reference discloses a utility infrastructure designed to span the globe and provide continuous access to persistent information. Because the infrastructure is comprised of untrusted servers, data is protected through redundancy and cryptographic techniques. To improve performance, data is allowed to be cached anywhere, anytime. Monitoring of usage patterns allows adaptation to regional outages and denial of service attacks. Monitoring also enhances performance through proactive movement of data.

(f) In view of this petition, the Examiner is respectfully requested to issue a first Office Action at an early date.

Respectfully submitted,

Chun-Pok Leung
Reg. No. 41,405

TOWNSEND and TOWNSEND and CREW LLP
Two Embarcadero Center, 8th Floor
San Francisco, California 94111-3834
Tel: 650-326-2400
Fax: 415-576-0300
Attachments
RL:rl
60272139 v1

FEE TRANSMITTAL for FY 2004

Effective 10/01/2003. Patent fees are subject to annual revision.

☐ Applicant claims small entity status. See 37 CFR 1.27

TOTAL AMOUNT OF PAYMENT | ($) 130.00

Complete if Known	
Application Number	10/637,216
Filing Date	August 8, 2003
First Named Inventor	SONODA, Koji
Examiner Name	Unassigned
Art Unit	2131
Attorney Docket No.	16869S-089800US

METHOD OF PAYMENT *(check all that apply)*

☐ Check ☐ Credit Card ☐ Money Order ☐ Other ☐ None

☒ Deposit Account:

Deposit Account Number: 20-1430

Deposit Account Name: Townsend and Townsend and Crew LLP

The Director is authorized to: *(check all that apply)*

☒ Charge fee(s) indicated below ☒ Credit any overpayments

☒ Charge any additional fee(s) or any underpayment of fee(s)

☐ Charge fee(s) indicated below, **except for the filing fee** to the above-identified deposit account.

FEE CALCULATION

1. BASIC FILING FEE

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
1001	770	2001	385	Utility filing fee	
1002	340	2002	170	Design filing fee	
1003	530	2003	265	Plant filing fee	
1004	770	2004	385	Reissue filing fee	
1005	160	2005	80	Provisional filing fee	

SUBTOTAL (1) ($)0.00

2. EXTRA CLAIM FEES FOR UTILITY AND REISSUE

			Extra Claims		Fee from below		Fee Paid
Total Claims		-** =		X		=	
Independent Claims		-** =		X		=	
Multiple Dependent				X		=	

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description
1202	18	2202	9	Claims in excess of 20
1201	86	2201	43	Independent claims in excess of 3
1203	290	2203	145	Multiple dependent claim, if not paid
1204	86	2204	43	** Reissue independent claims over original patent
1205	18	2205	9	** Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) ($)0.00

**or number previously paid, if greater; For Reissues, see above*

FEE CALCULATION (continued)

3. ADDITIONAL FEES

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
1051	130	2051	65	Surcharge - late filing fee or oath	
1052	50	2052	25	Surcharge - late provisional filing fee or cover sheet.	
1053	130	1053	130	Non-English specification	
1812	2,520	1812	2,520	For filing a request for reexamination	
1804	920*	1804	920*	Requesting publication of SIR prior to Examiner action	
1805	1,840*	1805	1,840*	Requesting publication of SIR after Examiner action	
1251	110	2251	55	Extension for reply within first month	
1252	420	2252	210	Extension for reply within second month	
1253	950	2253	475	Extension for reply within third month	
1254	1,480	2254	740	Extension for reply within fourth month	
1255	2,010	2255	1,005	Extension for reply within fifth month	
1401	330	2401	165	Notice of Appeal	
1402	330	2402	165	Filing a brief in support of an appeal	
1403	290	2403	145	Request for oral hearing	
1451	1,510	1451	1,510	Petition to institute a public use proceeding	
1452	110	2452	55	Petition to revive – unavoidable	
1453	1,330	2453	665	Petition to revive – unintentional	
1501	1,330	2501	665	Utility issue fee (or reissue)	
1502	480	2502	240	Design issue fee	
1503	640	2503	320	Plant issue fee	
1460	130	1460	130	Petitions to the Commissioner	130
1807	50	1807	50	Petitions related to provisional applications	
1806	180	1806	180	Submission of Information Disclosure Stmt	
8021	40	8021	40	Recording each patent assignment per property (times number of properties)	
1809	770	2809	385	Filing a submission after final rejection (37 CFR § 1.129(a))	
1810	770	2810	385	For each additional invention to be examined (37 CFR § 1.129(b))	
1801	770	2801	385	Request for Continued Examination (RCE)	
1802	900	1802	900	Request for expedited examination of a design application	
Other fee (specify) ----------					

*Reduced by Basic Filing Fee Paid **SUBTOTAL (3)** ($)130.00

SUBMITTED BY				Complete *(if applicable)*	
Name (Print/Type)	Chun-Pok Leung	*Registration No. (Attorney/Agent)*	41,405	*Telephone*	650-326-2400
Signature				*Date*	August 6, 2004

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

60278906 v1

3403000222

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2003-015931

(43)Date of publication of application : 17.01.2003

(51)Int.Cl. G06F 12/00

(21)Application number : 2001-200455

(22)Date of filing : 02.07.2001

(71)Applicant : HITACHI LTD

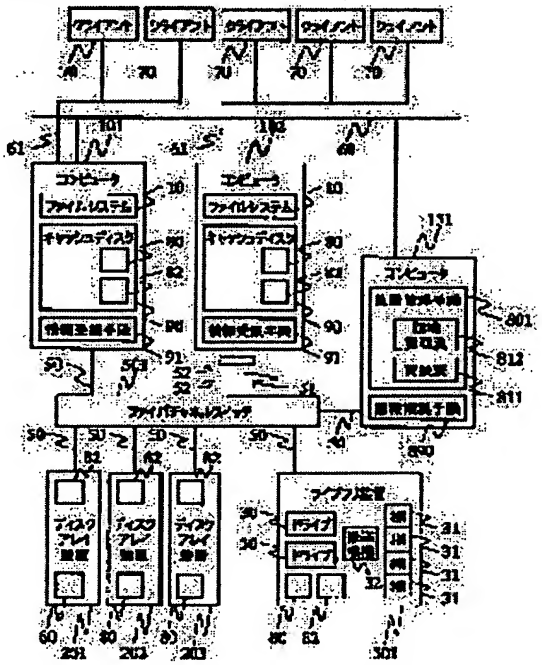
(72)Inventor : KANEDA TAISUKE
ARAKAWA TAKASHI
EGUCHI KENTETSU
MOGI KAZUHIKO
ARAI HIROHARU

(54) INFORMATION PROCESSING SYSTEM AND STORAGE AREA PROVIDING METHOD

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a data storage system in consideration of the attributes (performance and cost), the operating rate and the data usage frequency of the storage device in a storage area network constituted of a plurality of the storage devices that have different attributes and are used in a business pattern of a storage service provider or the like.

SOLUTION: The information processing system is provided with a location management means for managing the location of data stored in the storage device, an information replication means for replicating data between the storage devices, and an attribute holding means for holding information representing the attribute of each storage device. The location management means is so constituted as to use the information replication means to replicate and move the data between the storage devices based on the operating rate of, the data usage status of, control information on and the accounting information on the storage device.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2003－15931
（P2003－15931A）
(43)公開日　平成15年１月17日(2003.1.17)

(51)Int.Cl.[7]	識別記号	ＦＩ	ﾃｰﾏｺｰﾄﾞ(参考)
Ｇ０６Ｆ　12/00	５４５	Ｇ０６Ｆ　12/00	５４５Ａ　５Ｂ０８２
	５２０		５２０Ｅ
			５２０Ｊ

審査請求　未請求　請求項の数16　ＯＬ　（全 27 頁）

(21)出願番号　特願2001－200455(P2001－200455)

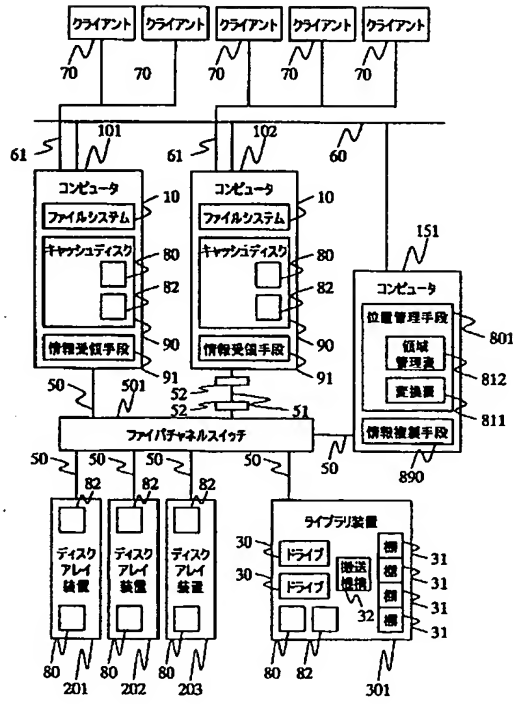
(22)出願日　平成13年７月２日(2001.7.2)

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地
(72)発明者　兼田　泰典
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内
(72)発明者　荒川　敬史
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内
(74)代理人　100096954
弁理士　矢島　保夫

最終頁に続く

(54)【発明の名称】　情報処理システムおよび記憶領域提供方法

(57)【要約】
【課題】ストレージサービスプロバイダなどのビジネス形態で利用する、複数の異なる属性を有する記憶装置で構成されるストレージエリアネットワークにおいて、記憶装置の属性（性能やコスト）や稼働率およびデータの利用頻度を考慮したデータの保持方式を提供することを目的とする。
【解決手段】記憶装置に格納されたデータの位置を管理する位置管理手段と、記憶装置間でデータを複製する情報複製手段と、各記憶装置の属性を表す情報を保持する属性保持手段を設ける。位置管理手段は、記憶装置の稼働率や、データの利用状況、制御情報、課金情報をもとに、情報複製手段を用いて、記憶装置間のデータの複製および移動を行う構成とする。



【特許請求の範囲】

【請求項１】情報を記憶する複数の記憶装置と、前記記憶装置に情報を書き込みまたは前記記憶装置から情報を読み出す１つまたは複数の制御装置とを備えた情報処理システムにおいて、

前記記憶装置に格納された情報の位置を管理する位置管理手段と、前記記憶装置間で情報の複製を行う情報複製手段と、前記各記憶装置の属性を表す属性情報を保持する属性保持手段とを備え、

前記制御装置が前記記憶装置に情報を書き込んだり、前記記憶装置から情報を読み出したりする場合、前記制御装置は前記位置管理手段から指示された位置情報に基づいて書き込みまたは読み出しの処理を行い、

前記位置管理手段は、前記属性保持手段の属性情報を用いて、前記制御装置が前記記憶装置に書き込もうとする情報または書き込まれた情報の位置管理を行うことを特徴とする情報処理システム。

【請求項２】請求項１に記載の情報処理システムにおいて、前記属性保持手段は前記各記憶装置内に設けたことを特徴とする情報処理システム。

【請求項３】請求項１に記載の情報処理システムにおいて、前記位置管理手段は、前記制御装置から参照される情報識別子に対応した情報位置を少なくとも二つ以上保持可能とすることを特徴とする情報処理システム。

【請求項４】請求項１に記載の情報処理システムにおいて、前記記憶装置の稼働率を計測する稼働率計測手段を設け、前記位置管理手段は、前記属性保持手段の情報と、前記稼働率保持手段に保持された前記記憶装置の稼働率に基づいて、前記制御装置が前記記憶装置に書き込もうとする情報または書き込まれた情報の位置管理を行うことを特徴とする情報処理システム。

【請求項５】請求項１に記載の情報処理システムにおいて、前記記憶装置の稼働率を計測する稼働率計測手段を設け、前記位置管理手段は、前記属性保持手段の情報と、前記稼働率保持手段に保持された前記記憶装置の稼働率に基づいて、前記制御装置が前記記憶装置から読み出そうとする情報の位置管理を行うことを特徴とする情報処理システム。

【請求項６】請求項１に記載の情報処理システムにおいて、前記位置管理手段は、前記制御装置から参照される情報識別子に対応して当該情報の利用状況を保持し、前記情報複製手段を用いて、当該情報を複製または移動することを特徴とする情報処理システム。

【請求項７】請求項６に記載の情報処理システムにおいて、前記位置管理手段は、前記制御装置から参照される情報識別子に対応して当該情報の複製や移動を制御するための制御情報を保持し、当該情報の利用状況に加えて、前記制御装置が設定した制御情報をも用いて当該情報を複製または移動することを特徴とする情報処理システム。

【請求項８】請求項７に記載の情報処理システムにおいて、前記記憶装置に関連付けられた制御情報を保持する記憶装置制御情報保持手段を設け、前記制御装置が制御情報を設定しなかった場合には、前記記憶装置制御情報保持手段に設定されている制御情報を用いて情報の制御情報を設定し、前記位置管理手段は、利用状況に加えて、前記記憶装置制御情報保持手段より設定した制御情報を用いて、当該情報を複製または移動することを特徴とする情報処理システム。

【請求項９】請求項１に記載の情報処理システムにおいて、前記位置管理手段は、前記制御装置から参照される情報識別子に対応して当該情報の移動、および複製履歴を保持することを特徴とする情報処理システム。

【請求項１０】請求項１に記載の情報処理システムにおいて、前記属性保持手段は、情報の保持にかかるコストに関するコスト情報を保持することを特徴とする情報処理システム。

【請求項１１】請求項１０に記載の情報処理システムにおいて、前記位置管理手段は、利用状況、前記記憶装置制御情報保持手段より設定した制御情報に加えて、前記コスト情報を用いて情報を複製または移動することを特徴とする情報処理システム。

【請求項１２】請求項１に記載の情報処理システムにおいて、前記位置管理手段は、前記制御装置から参照される情報識別子に対応して当該情報のグループ識別子を保持し、該グループ識別子を用いて情報を複製または移動することを特徴とする情報処理システム。

【請求項１３】請求項１に記載の情報処理システムにおいて、前記制御装置は、前記情報複製手段が転送する情報を受け取るための情報受領手段と、受け取った情報を保持する記憶手段を設けたことを特徴とする情報処理システム。

【請求項１４】請求項１に記載の情報処理システムにおいて、前記情報複製手段は、複製元の一つの情報を、複製先の複数の領域へ複製可能なことを特徴とする情報処理システム。

【請求項１５】記憶装置の記憶領域を複数のユーザに提供する記憶領域提供方法であって、

ネットワークを介して、ユーザから、保存すべきデータと該データをどのように保存するかを指定する制御情報とを受け取るステップと、

前記制御情報に応じた属性を持つ記憶装置に前記データを保存するステップとを備えたことを特徴とする記憶領域提供方法。

【請求項１６】記憶装置の記憶領域を複数のユーザに提供する記憶領域提供方法であって、

ネットワークを介して、ユーザから、保存すべきデータと該データをどのように保存するかを指定する制御情報とを受け取るステップと、

前記制御情報に応じた属性を持つ記憶装置に前記データ

を保存するステップと、
前記データを保存した記憶装置の単位容量あたりの保持コストと、前記データを保存した期間に応じて、前記ユーザへの課金を行なうステップとを備えたことを特徴とする記憶領域提供方法。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、情報処理システムにおける記憶装置（ディスクアレイ装置やハードディスク装置、およびテープライブラリ装置や光ディスクライブラリ装置など）の利用技術に関し、特に、複数の異なる特性をもつ記憶装置を複数の制御装置（コンピュータ）から利用する技術に関する。
【０００２】
【従来の技術】複数のコンピュータ（制御装置）がストレージ（記憶装置）を共有する技術として、ストレージエリアネットワークが広く使われるようになってきた。ストレージエリアネットワークが利用される以前は、コンピュータシステムのファイルシステムソフトウエアは、複数のコンピュータで一つのストレージを共有して使用することが考慮されておらず、ストレージエリアネットワーク環境で共有して利用すると、ストレージに記録されているデータ（情報）が失われてしまう危険性があった。この点に関しては、米国特許第5,950,203号にその記載がある。
【０００３】上記特許では、複数のコンピュータからストレージを共有する技術が開示されているが、ストレージが異なる属性（性能、信頼性、保存性、容量あたりのコストなど）を有する場合のストレージの利用方式についての記載がない。
【０００４】この点については、Tivoli Systems Inc.のホワイトペーパー「Vision, Tivoli Storage Management Solutions for the Information Grid」に、ストレージエリアネットワークにおいて、属性の異なるストレージを使い分けてデータを保持する技術が紹介されている。しかし、このホワイトペーパにおいても、ストレージにおける属性管理の方法や位置管理手段におけるデータ管理の方式、およびデータの保持に関する課金方式などについては記載がない。
【０００５】一方、ストレージサービスプロバイダ（SSP:Storage Service Provider）と呼ばれるビジネスがある。ストレージサービスプロバイダは、顧客の要求にあったストレージをアウトソーシングにより提供する。顧客はストレージサービスプロバイダにストレージの管理を委託することで、ストレージの維持管理コストを削減できる。サービスプロバイダは容量の管理を行えばよいだけではなく、顧客の求める性能や信頼性、コスト、耐障害性などに答えていかなければならない。また、ストレージサービスプロバイダ自身も、性能や信頼性を維持しつつ、ストレージにかかる維持管理コストを削減しなければならない。
【０００６】
【発明が解決しようとする課題】本発明の目的は、たとえばストレージサービスプロバイダなどのビジネス形態で実施される、複数の異なる属性を有するストレージ（記憶装置）で構成されるストレージエリアネットワークにおいて、ストレージの属性（性能やコスト）や稼働率、およびデータの利用頻度を考慮したデータの保持方式を提供することにある。
【０００７】
【課題を解決するための手段】上記目的を達成するために、本発明は、記憶装置に格納されたデータの位置を管理する位置管理手段と、記憶装置間でデータの複製を行う情報複製手段と、各記憶装置の属性を表す属性情報を保持する属性保持手段とを設けた。この属性保持手段は、各記憶装置毎に設けても良い。この属性保持手段に保持した属性情報に基づき、位置管理手段はデータの書き込みおよび読み込みのための位置管理を行う。
【０００８】また、位置管理手段には、コンピュータから示されるファイルネーム（情報の存在場所を示す情報識別子）に対応するデータの位置情報を少なくとも二つ以上保持できるように構成した。これにより、一つのファイルネームに対して、複数のデータを異なるストレージに保存することができるようになる。
【０００９】また、ストレージの稼働率を計測する稼働率計測手段を設け、位置管理手段はこの稼働率をもとに、書き込もうとするデータの位置管理や、書き込まれたデータの位置管理、読み出すデータの位置管理を行う構成とした。この稼働率計測手段は各ストレージ毎に設けても良い。
【００１０】また、位置管理手段には、コンピュータから示されるファイルネームに対応して、そのデータの利用状況と、そのデータに対する制御情報（ユーザが設定したデータの複製、および移動のルール）を保持できる構成にした。制御情報は、各データ毎にコンピュータから設定することもできるし、あらかじめ位置管理手段に設定しておいた制御情報を用いることもできる。
【００１１】また、位置管理手段には、コンピュータから示されるファイルネームに対応して、そのデータの移動および複製履歴を保持できる構成にした。さらに、属性保持手段にコスト情報を保持する構成にもできる。これにより、位置管理手段は、ストレージの稼働率、データの利用状況、制御情報、あるいはコスト情報をもとに、情報複製手段を用いて、記憶装置間のデータの複製および移動を行える構成となる。
【００１２】また、位置管理手段には、コンピュータから示されるファイルネームに対応して、そのデータのグループＩＤを保持できる構成にし、位置管理手段は、このグループＩＤに基づきデータの複製、および移動を行える構成とした。

【００１３】また、コンピュータには、情報複製手段が転送する情報を受け取るための情報受領手段と、受け取った情報を保持する記憶手段を設けた。

【００１４】

【発明の実施の形態】以下、図面を用いて本発明の第１の実施形態（キャッシュなしの場合）を説明する。

【００１５】図１は、第１の実施形態のシステム構成図を示す。本システムは、３台のコンピュータ１０１，１０２，１５１と、３台のディスクアレイ装置２０１，２０２，２０３と、ライブラリ装置３０１と、ファイバチャネルスイッチ５０１とを備えている。コンピュータ１０１，１０２，１５１と、ディスクアレイ装置２０１，２０２，２０３と、ライブラリ装置３０１は、ファイバチャネル５０を介して、ファイバチャネルスイッチ５０１に接続する。ファイバチャネルスイッチ５０１は、接続された各機器間のパスの確立と切り替えを行う。また、コンピュータ１０１，１０２，１５１は、イーサーネット６０を用いて相互に接続する。

【００１６】なお、第１の実施形態の後に第２および第３の実施形態について説明するが、第２および第３の実施形態のシステムも第１の実施形態のシステムと同様の構成であるので図１を用いて説明する。図１において、コンピュータ１０１，１０２中のキャッシュディスク９０と情報受領手段９１、並びに、コンピュータ１０１，１０２、ディスクアレイ装置２０１，２０２，２０３、およびライブラリ装置３０１中の稼働率計測手段８２については、第２の実施形態のシステムに固有の構成であるので第１の実施形態では言及しない。同様に、クライアント７０についても、第３の実施形態のシステムに固有の構成であるので第１および第２の実施形態では言及しない。

【００１７】図１において、ファイバチャネル５０を用いて構成したネットワークは一般的にＳＡＮ(Storage Area Network)と呼ばれ、イーサーネット６０を用いて構成したネットワークは一般的にＬＡＮ(Local Area Network)と呼ばれる。図１では、ファイバチャネル５０とイーサネット６０の二つのネットワークを用いて構成したが、これらは、他のネットワークでもよいし、イーサネット６０の機能をファイバチャネル５０で肩代わりすることでも実現可能である。

【００１８】３台のコンピュータのうち、コンピュータ１５１は、データの位置管理手段８０１と情報複製手段８９０を備える。位置管理手段８０１と情報複製手段８９０の詳細については後述する。

【００１９】次に、ディスクアレイ装置と属性保持手段について説明する。ディスクアレイ装置は、一般的に複数のハードディスク装置を有し、ＲＡＩＤ(Redundant Arrayof Inexpensive Disks)技術により、性能および信頼性の改善を図った記憶装置である。ディスクアレイ装置は、ディスクアレイ装置を構成するハードディスク装置の性能、台数、およびＲＡＩＤレベルによりその性能と信頼性が大きく異なってくる。たとえば、ＲＡＩＤ０と呼ばれるレベルは、データを複数のハードディスクに分散することで性能は高くなるが、ＲＡＩＤ０を構成する複数のハードディスクのうち１台でも故障するとデータを失う危険性がある。また、ＲＡＩＤ５と呼ばれるレベルは、データを複数のハードディスクに分散することで性能を上げ、さらにパリティによる冗長データを付加することで、ＲＡＩＤ５を構成する複数のハードディスクのうち１台が故障しても、データを復元可能な信頼性を提供している。ＲＡＩＤ５は、ＲＡＩＤ０に比べてパリティによる信頼性向上が見込まれるが、その分、書き込み時の性能が低下する。また、ＲＡＩＤ０およびＲＡＩＤ５とも、構成するハードディスク装置の台数によりその性能が異なる。

【００２０】本実施形態では、ディスクアレイ装置２０１と２０２をＲＡＩＤ０で構成し、ディスクアレイ装置２０３をＲＡＩＤ５で構成するものとする。それぞれのディスクアレイ装置には、属性保持手段８０を設けた。ディスクアレイ装置２０１と２０２の属性保持手段８０は、ハードディスク装置で構成されるＲＡＩＤ０であること、単位容量あたりの保持コスト、構成するハードディスク装置台数、ハードディスク装置の性能（回転数、シーク時間、転送速度など）を保持する。ディスクアレイ装置２０３の属性保持手段８０は、ハードディスク装置で構成されるＲＡＩＤ５であること、単位容量あたりの保持コスト、構成するハードディスク装置台数、ハードディスク装置の性能（回転数、シーク時間、転送速度など）を保持する。よって、位置管理手段８０１は、属性保持手段８０からこれらの属性情報を取り出せば、そのディスクアレイ装置がどのような性能と信頼性を有するかを容易に把握することができる。属性保持手段８０の実現手段としては、ディスクアレイ装置内にある不揮発性の記憶手段が望ましい。たとえば、装置識別子や装置情報が格納されているメモリを用いて属性保持手段８０を実現すればよい。また、属性保持手段８０に格納されている情報の取出しには、ＳＣＳＩ(SmallComputer System Interface)で定義されているＩＮＱＵＩＲＹコマンドやＭＯＤＥ　ＳＥＮＳＥコマンドを利用する。

【００２１】次に、ライブラリ装置と属性保持手段について説明する。本実施形態において、ライブラリ装置３０１は、２台のＤＶＤ－ＲＡＭドライブ３０と、メディアを格納可能な４個の棚３１と、メディアをドライブ３０と棚３１との間で搬送する１つの搬送機構３２を有する。ただし本発明は、ライブラリ装置におけるドライブ台数、棚の数、搬送機構の数などに依存しない。また、本発明は、複数のライブラリ装置を用いても構成できる。本実施形態では、４個の棚３１それぞれに１枚ずつＤＶＤ－ＲＡＭメディアを格納するものとする。ライブラリ装置３０１は、搬送機構３２を用いて、コンピュー

タからの指示に基づいて、指定されたメディアを、ドライブードライブ間、ドライブー棚間、および棚ー棚間で搬送することができる。

【００２２】ライブラリ装置３０１に格納されたメディアに記録されているデータを読み出すためには、コンピュータは、まずライブラリ装置に、読みたいデータの記録されているメディアを棚からドライブに搬送するように指示する。コンピュータは、ライブラリ装置３０１からの搬送終了の報告を待って、ドライブ３０にデータの読み出しを指示する。このように、目的のデータを読み出すためには、搬送機構３２によるメディアの搬送時間と、ドライブにおいてメディアが読み出し可能になるまでの時間（主にメディアが読み出し可能な回転数に達するまでのスピンアップ時間）がかかることになる。この時間を約１０秒とすると、ディスクアレイ装置２０１，２０２，２０３などのデータ読み出しにかかる時間が数ｍｓであることと比べると、その差は3桁以上になる。

【００２３】本実施形態では、ライブラリ装置３０１にも属性保持手段８０を設けた。ライブラリ装置３０１の属性保持手段８０は、ＤＶＤ－ＲＡＭドライブで構成されるライブラリ装置であること、単位容量あたりの保持コスト、搬送時間、ライブラリ装置３０１に搭載されているドライブ３０の台数および種類、メディアの枚数、種類および寿命、並びに、ドライブの性能（回転数、シーク時間、転送速度など）を保持する。属性保持手段８０は、ライブラリ装置３０１内にある不揮発性の記憶手段を用いて実現するのが望ましい。

【００２４】次に、位置管理手段と変換表について説明する。

【００２５】コンピュータ１５１は、位置管理手段８０１を備える。具体的には、位置管理手段８０１は、コンピュータ１５１上で実行されるソフトウエアとして実装してある。位置管理手段８０１は、コンピュータ１０１や１０２とはＬＡＮ６０を介して通信する。通信のためのプロトコルに関しては特に限定されることはないが、たとえばＴＣＰやＵＤＰなどが利用できる。位置管理手段８０１は、コンピュータ１０１や１０２からファイルネームを受け取ると、そのファイルネームのファイルのデータを必要に応じてロックし、そのデータの位置をコンピュータ１５１に報告する。このため、ファイルネームとデータの位置情報とを対応付けた変換表８１１を用意する。

【００２６】図２は、変換表８１１の構成を示す。変換表８１１は、ホストコンピュータ１５１のメモリに格納される。変換表８１１は、１つのファイルネームに対してライトビット、リードカウント、制御情報、複数の位置情報、およびファイル履歴の各フィールドを備える（グループＩＤについては後述する）。また、各位置情報は、有効ビット、記憶装置番号、メディア番号、ＬＢＡ(Logical Block Address)、およびデータ長を持つことができる。本実施形態の場合、記憶装置番号には、WWN(World Wide Name)を使用した。WWNは、ファイバチャネルアダプタ固有に割り当てられる番号である。本実施形態では、ディスクアレイ装置２０１のWWNを「２０１」、ディスクアレイ装置２０２のWWNを「２０２」、ディスクアレイ装置２０３のWWNを「２０３」、ライブラリ装置３０１のWWNを「３０１」とした。メディア番号は、対象となる記憶装置がライブラリ装置の場合にのみ有効で、そのデータが格納されているメディアの番号（または棚の番号）を保持する。ＬＢＡには、そのデータの格納されている論理ブロックアドレスを使用する。論理ブロックとは、記憶装置が管理するデータを記憶する最小単位である。ディスクアレイの場合には一般的に５１２バイトのデータを格納でき、ＤＶＤ－ＲＡＭメディアの場合には一般的に２０４８バイトのデータを格納できる。論理ブロックアドレスとは、この論理ブロックに割り当てられた番号のことである。位置情報の有効ビットがセットされている場合に限り、これら位置情報が有効になる。

【００２７】次に、位置管理手段と領域管理表について説明する。位置管理手段８０１には、領域管理表８１２を設けた。図３に領域管理表８１２の一例を示す。領域管理表８１２は、ディスクアレイ装置２０１，２０２，２０３の場合にはディスクアレイ装置毎に、ライブラリ装置３０１の場合にはメディア毎に用意する。領域管理表８１２は、その装置またはメディアにおいてどの論理ブロックが使われているかを示した表で、１つの論理ブロックあたり１ビットの情報で構成している。すなわち、論理ブロックに対応したビットが「1」の場合にはすでにその論理ブロックにデータがあり、「0」の場合には未使用であることを示す。領域管理表８１２は、コンピュータ１５１のメモリに格納される。

【００２８】次に、新規書き込み処理について説明する。図５は、本システムにおいて、記憶装置にデータを新規に書き込む処理手順を示すフローチャートである。コンピュータ１０１を使用するユーザが、ワープロなどのアプリケーションを用いて新規に文書を作成し初めて保存する場合などに相当する（５０００）。

【００２９】コンピュータ１０１のファイルシステム１０は、イーサネット６０を介して、ファイルネーム「ＡＡＡ」というデータに対する新規領域割り当てを、コンピュータ１５１の位置管理手段８０１に要求する（５００１）。このとき、コンピュータ１０１は、データ「ＡＡＡ」（データ「ＡＡＡ」とは、ファイルネーム「ＡＡＡ」のファイルの実体データである）のデータ長に加えて、データ「ＡＡＡ」をどのように保存したいかを示す制御情報を数値化して付加する。制御情報の例としては、「高速に」、「高信頼に」、「長期に」、「低価格に」、「災害対応に」、「一時的に」、「並列に」、および「可搬に」などがある。これらは同時に設定可能と

してもよいし、排他的な設定とすることもできる。ここでは、「高信頼に」を指定したとする。

【００３０】位置管理手段８０１は、まず変換表８１１から空いている一行を探し出し（５００２）、ライトビットを「1」にして（５００３）書き込みの開始を宣言し、ファイルネームを書き込む（５００４）。ここでは変換表８１１の１番目（＃１）の位置情報を確保したとする。領域管理表８１２から、データ「ＡＡＡ」を格納するのに必要な領域を検索する（５００５）。ここでは、すべての記憶装置に十分な空き領域があるとすると（５００６，５００７）、位置管理手段８０１は、すべての記憶装置から割り当て可能位置を獲得することができる。位置管理手段８０１は、それぞれの記憶装置の属性保持手段８０から、その記憶装置の属性情報を獲得する（５００８）。位置管理手段８０１は、コンピュータ１０１が指示した「高信頼に」という制御情報をもとに、３台のディスクアレイ装置の中から、ＲＡＩＤ５で構成されたディスクアレイ装置２０３を選択し（５００９）、対応する領域管理表８１２のＬＢＡを割り当て済みに変更し（５０１０）、位置情報をコンピュータ１０１に報告する（５０１１）。

【００３１】コンピュータ１０１のファイルシステム１０は、空き領域の報告を受けると、指定された空き領域にデータを書き込むためのライトコマンドをディスクアレイ２０３に発行し（５０１２）、データを書き込む（５０１４）。このライトコマンドの発行と書き込むデータの転送は、ファイバチャネル５０を介して行う。コンピュータ１０１のファイルシステム１０は、ディスクアレイ装置２０３から書き込み完了の報告（５０１５）を受け取ると、コンピュータ１５１の位置管理手段８０１に書き込みが完了したことを報告する（５０１６）。

【００３２】位置管理手段８０１は、書き込み完了の報告を受け取ると、変換表８１１の当該位置情報、ここでは１番目の位置情報に、ディスクアレイ装置２０３のＷＷＮ、ＬＢＡ、およびデータ長を書き込み（５０１７）、有効ビットを「1」にセットする（５０１８）ことで、１番目の位置情報が有効であることを示す。この場合、データをディスクアレイ装置に書き込んだので、メディア番号には値を入れる必要はない。また、ファイル履歴には、新規作成とその日時を書き込む（５０１９）。最後に、ライトビットを「0」にして（５０２０）、書き込みを完了する（５０２１）。ファイルシステム１０は、位置管理手段８０１から書き込み完了の報告を受け取ると、アプリケーションに書き込み完了の報告をする（５０２２）。これにより、アプリケーションからの新規文書保存が完了する（５０２３）。

【００３３】図５に示すフローチャートの処理ステップ５００６において、割り当て可能な領域が無かった場合には、書き込み処理は失敗となる（５０９８，５０９９）。また処理ステップ５００６で割り当て可能な領域はあるが、処理ステップ５００７で複数の記憶装置に領域が無かったときは、処理ステップ５０１０に進む。

【００３４】次に、読み出し処理について説明する。図６は、本システムにおいて記憶装置からデータを読み出す処理手順を示すフローチャートである。コンピュータ１０１を使用するユーザが、ワープロなどのアプリケーションを用いて、前回保存しておいた文書を再読み込みする場合などに相当する（６０００）。

【００３５】コンピュータ１０１のファイルシステム１０は、イーサネット６０を介して、ファイルネーム「ＡＡＡ」のデータの読み出しを位置管理手段８０１に要求する（６００１）。位置管理手段８０１は、変換表８１１からファイルネーム「ＡＡＡ」に対応する行を選択し（６００２）、ライトビットが「0」であることを確認する（６００３）。もしライトビットが「1」の場合には、他のアプリケーション（他のコンピュータのアプリケーションでもよい）がデータ「ＡＡＡ」を利用中であるので読み出しをすることができない（６０９８，６０９９）。

【００３６】次に、リードカウントをインクリメント（1加算）し（６００４）、データ「ＡＡＡ」の入っている位置情報をコンピュータ１０１に報告する（６００７）。データ「ＡＡＡ」に対応する位置情報を１つしかもっていない場合には、その位置情報を報告するしかないが、もし、２つ以上の位置情報を有する場合には、コンピュータ１０１に最適な（たとえば、どこから読み出すのが最も速いかなど）位置情報を選択し（６００６）、報告する（６００７）。ここでは、上記新規書き込みで書き込んだデータ「ＡＡＡ」の位置情報であるとする。

【００３７】コンピュータ１０１のファイルシステム１０は、位置情報（ＷＷＮ、ＬＢＡ、データ長）の報告を受けると、ディスクアレイ装置２０３にリードコマンドを発行し（６００８）、データを読み出す（６００９）。このリードコマンドの発行とリードデータの転送は、ファイバチャネル５０を介して行う。コンピュータ１０１のファイルシステム１０は、ディスクアレイ装置２０３から読み出し完了の報告（６０１０）を受け取ると、コンピュータ１５１の位置管理手段８０１に読み出しが完了したことを報告する（６０１１）。

【００３８】位置管理手段８０１は、読み出し完了の報告を受け取ると、ファイル履歴に最終アクセス日時を書き込み（６０１２）、リードカウントをデクリメント（1減算）する（６０１３）。リードカウントが０であれば、誰もデータ「ＡＡＡ」を参照していないことを示し、０でなければまだ読み出し中であることがわかる。ファイルシステム１０は、位置管理手段８０１から読み出し完了の報告（６０１４）を受け取ると、アプリケーションに読み出しの完了を報告する（６０１５）。これにより、アプリケーションからの文書読み出しが完了す

る（６０１６）。

【００３９】次に、制御情報の変更について説明する。ユーザは、コンピュータ１０１や１０２上に用意したユーティリティを用いて、設定した制御情報の参照および再設定が可能である。ここでは、先に書き込んだデータ「ＡＡＡ」の制御情報を変更する例を示す。ユーザが、制御情報の再設定をユーティリティに要求すると、ファイルシステム１０は、位置管理手段８０１に制御情報の読み出しを要求する。位置管理手段８０１は、変換表８１１よりデータ「ＡＡＡ」の制御情報を読み出し、ファイルシステム１０に報告する。ファイルシステム１０は、制御情報を受け取ると、ユーティリティに対して制御情報を報告する。ここでは、位置管理手段８０１が、先に設定した「高信頼に」の制御情報を読み出したとする。

【００４０】次に、ユーザが「高信頼に」の制御情報を「低価格に」に変更すると、ファイルシステム１０は、位置管理手段８０１に制御情報の書き込みを要求する。位置管理手段８０１は、変換表８１１のデータ「ＡＡＡ」に対応する制御情報を「低価格に」に変更し、変更完了をファイルシステム１０に報告する。ファイルシステム１０は、ユーティリティに変更の完了を報告し、制御情報の変更が終了する。

【００４１】次に、位置管理手段８０１による監視と移動について説明する。図７と図８は、位置管理手段８０１におけるデータの監視と移動の処理手順を示すフローチャートである。ここでは、データ「ＡＡＡ」をディスクアレイ装置２０３からライブラリ装置３０１に移動する例について説明する。位置管理手段８０１は、一定の周期で変換表８１１を繰り返し検索している（７０００）。先ほど、制御情報を「高信頼に」から「低価格に」へ変更したデータ「ＡＡＡ」についても、この一定周期の巡回で変更が発見される（７００１）。位置管理手段８０１は、このデータの最終アクセス日時と現在の日時とを比較し（７００２）、一定期間（たとえば９０日とか）経っているか否かをチェックする（７００３）。一定期間経っていた場合には、移動対象であるものとして、データ「ＡＡＡ」をよりビットコストの安い記憶装置に移動する。本実施形態では、ディスクアレイ装置２０１と２０２の保持コストを「１０」、ディスクアレイ装置２０３の保持コストを「１５」、ライブラリ装置３０１の保持コストを「１」とすると、位置管理手段８０１は、ライブラリ装置３０１にデータ「ＡＡＡ」を移動することでデータの保持コストの低減を図る。

【００４２】そこで、位置管理手段８０１は、変換表８１１で、ライトビットが「０」で（７００４）、リードカウントが「０」であることを確認し（７００５）、ライトビットを「１」に設定し（７００６）、次に領域管理表８１２を参照しライブラリ装置３０１のメディアからデータ「ＡＡＡ」を格納するのに必要な領域を検索する。本実施形態の場合は、属性保持手段８０の属性情報を獲得し（７００７）、その情報をもとにデータ保持コストのより低い記憶装置を選択し（７００８）、新規領域を割り当てる（７００９）。

【００４３】データ「ＡＡＡ」を格納できる空き領域が見つかると（７０１０）、位置管理手段８０１は、対応する領域管理表８１２のＬＢＡを割り当て済みに変更し（７０１１）、情報複製手段８９０にデータ「ＡＡＡ」の複製を要求する（８０００）。本実施形態では、情報複製手段８９０を、コンピュータ１５１に設けているが、ファイバチャネルスイッチ５０１に設けてもよいし、ファイバチャネルスイッチ５０１に接続される他の機器に設けてもよい。

【００４４】図８に、情報複製手段８９０による複製処理の手順を示す。情報複製手段８９０は、まず、ライブラリ装置３０１に対して、空き領域が見つかったメディアをドライブ３０に搬送するように搬送命令を発行する（８００２）。すでに対象のメディアがドライブにある場合には、メディアを搬送する必要はない（８００１）。搬送機構３２は、指定されたメディアを、それが格納された棚から指定された目的のドライブへ搬送する（８００３，８００４）。情報複製手段８９０は、搬送したドライブで読み書きができる状態になるまで待つ（８００５）。次に、情報複製手段８９０は、すでにデータ「ＡＡＡ」が書き込まれているディスクアレイ装置２０３に対してリードコマンドを発行し（８００６）、データを読み出し（８００７，８００８）、搬送されたメディアが装着されたドライブにライトコマンドを発行し（８００９）、データを書き込む（８０１０，８０１１）。情報複製手段８９０は、ライトコマンドを完了すると、位置管理手段８０１に複製が完了したことを報告する（８０１２）。

【００４５】再び図７に戻って、位置管理手段８０１は、複製完了の報告を受け取ると、変換表８１１の空いている行、ここでは２番目の位置情報に、ライブラリ装置３０１のWWN、メディア番号、ＬＢＡ、およびデータ長を書き込み（７０１３）、有効ビットを「１」にセットする（７０１４）。次に、位置管理手段８０１は、１番目の位置情報の有効ビットを「０」にすることで（７０１５）、１番目の位置情報を無効にし、１番目の位置情報に対応する領域管理表８１２のＬＢＡを未使用にする。最後に、位置管理手段８０１は、ファイル履歴に移動日時を書き込み、ライトビットを「０」にして移動を完了する（７０１６）。

【００４６】このデータ「ＡＡＡ」の移動の後、たとえば、コンピュータ１０２からデータ「ＡＡＡ」の読み出しが要求されたとしても、位置管理情報８０１は、ライブラリ３０１に格納されたデータ「ＡＡＡ」の位置を正しく報告することができる。これは、位置管理手段８０１が、変換表８１１の位置情報を一元的に管理している

ためである。データをディスクアレイ装置２０３からライブラリ装置３０１に移動したことを、コンピュータ１０１と１０２に報告する必要はない。

【００４７】また、もしデータ「ＡＡＡ」が、「低価格に」に加えて「高信頼に」の制御情報をもっていた場合、位置管理手段８０１は、２枚のメディアに対して空き領域を検索し、各々にデータを移動する。これにより、１枚のメディアが障害を発生して読めなくなったり、１枚のメディアを紛失した場合でもデータの読み出しが可能になり、「高信頼に」の制御ができる。

【００４８】次に、ライブラリ装置３０１に格納されたデータの参照について説明する。図１５と図１６は、ライブラリ装置３０１に格納されたデータ「ＡＡＡ」の読み出し処理手順を示すフローチャートである。たとえば、上記ディスクアレイ装置２０３からライブラリ装置３０１へのデータ「ＡＡＡ」の移動後、コンピュータ１０２が、データ「ＡＡＡ」の読み出しを要求したとする（１５０００）。コンピュータ１０２のファイルシステム１０は、イーサネット（登録商標）６０を介して、ファイルネーム「ＡＡＡ」というデータの読み出しを要求する（１５００１）。

【００４９】位置管理手段８０１は、変換表８１１からファイルネーム「ＡＡＡ」に対応する行を選択し（１５００２）、ライトビットが「０」であることを確認し（１５００３）、リードカウントをインクリメントし読み出し処理を開始する（１５００４）。位置管理手段８０１は、ディスクアレイ装置２０１にデータ「ＡＡＡ」を格納可能な空き領域を検索する（１５００５）。空き領域が見つかると、対応する領域管理表８１２のＬＢＡを割り当て済みに変更し（１５００６）、情報複製手段８９０に対しデータ「ＡＡＡ」をライブラリ装置３０１からディスクアレイ装置２０１に複製するように要求する（１６０００）。

【００５０】図１６に、情報複製手段８９０による複製処理の手順を示す。情報複製手段８９０は、データ「ＡＡＡ」の格納されたメディアをドライブ３０に搬送するようにライブラリ装置３０１に指示する（１６００２）。もし、データ「ＡＡＡ」の格納されているメディアがすでにドライブ内にあるのであれば搬送する必要はない（１６００１）。搬送機構３２は、指定されたメディアの格納された棚から指定された目的のドライブへメディアを搬送する（１６００３，１６００４）。ドライブに搬送したメディアが読み出し可能になるのを待って（１６００５）、ライブラリ装置３０１に対してリードコマンドを発行してデータを読み出し（１６００７，１６００８）、ディスクアレイ装置２０１にライトコマンドを発行し（１６００９）、データを書き込む（１６０１０，１６０１１）。情報複製手段８９０は、ライトコマンドを完了すると、位置管理手段８０１に複製が完了したことを報告する（１６０１２）。

【００５１】再び図１５に戻って、位置管理手段８０１は、複製完了の報告を受け取ると、変換表８１１の空いている位置情報にディスクアレイ装置２０１のWWN、ＬＢＡ、およびデータ長を書き込み（１５００７）、有効ビットを「１」にセットし（１５００８）、この位置情報をコンピュータ１０２のファイルシステム１０に報告する（１５００９）。

【００５２】コンピュータ１０２のファイルシステム１０は、位置情報（WWN、メディア番号、ＬＢＡ、データ長）の報告を受けると、ディスクアレイ装置２０１にリードコマンドを発行し（１５０１０）、データを読み出す（１５０１１，１５０１２）。コンピュータ１０２のファイルシステム１０は、ディスクアレイ装置２０１から読み出し完了の報告を受け取ると、コンピュータ１５１の位置管理手段８０１に読み出しが完了したことを報告する（１５０１３）。位置管理手段８０１は、読み出し完了の報告を受け取ると、変換表８１１のファイル履歴に最終アクセス日時と複製日時を書き込み（１５０１４）、リードカウントをデクリメントし（１５０１５）、読み出しが完了する（１５０１６，１５０１７，１５０１８）。

【００５３】この場合、たとえデータ「ＡＡＡ」に「高信頼に」の制御情報が与えられていたとしても、ＲＡＩＤ５で構成されているディスクアレイ２０３にファイル「ＡＡＡ」を複製する必要性は必ずしもない。なぜなら、ライブラリ装置３０１とＲＡＩＤ０のディスクアレイ装置２０１の２箇所にデータが保持されているためである。

【００５４】ここで、位置管理手段８０１は、複製したディスクアレイ装置２０１上のデータが一定期間アクセスされていないことを確認すると、変換表８１１の位置情報の有効ビットを「０」にリセットする。この際、この位置情報に対応する領域管理表８１２のＬＢＡは未使用に戻される。

【００５５】次に、制御情報のバリエーションについて説明する。これまでに「低価格に」と「高信頼に」の２つの制御情報について記載してきたが、その他の制御情報が設定された場合の動作について記載する。

【００５６】「高速に」の制御情報が与えられた場合には、２台のＲＡＩＤ０ディスクアレイ装置２０１と２０２の２箇所にデータを記録する。ＲＡＩＤ０は信頼性がＲＡＩＤ５に比べて劣るため、ＲＡＩＤ０のディスクアレイ装置２台にデータを記録することで信頼性を維持する。「高速に」の制御情報に加えて、「一時的に」の制御情報が付加されていた場合には、１台のＲＡＩＤ０ディスクアレイ装置にのみデータを記録する。

【００５７】「長期に」の制御情報が与えられた場合には、ＤＶＤ－ＲＡＭメディアなど、取り外し可能な光記録メディアにデータを記録する。ＤＶＤ－ＲＡＭメディアは、３０年の保存が可能であるといわれており、ハー

ドディスク装置に比べ長期間に渡りデータを保管するのに向いている。「長期に」の制御情報に加えて、「高信頼に」の制御情報が付加されている場合には、２枚のメディアにデータを記録する。また、「長期に」の制御情報に加えて、「高速に」の制御情報が付加されている場合には、ディスクアレイ装置にもデータを記録しておく。ディスクアレイ装置に記録したデータは、アクセスの遅いライブラリ装置に対するキャッシュとして働く。

【００５８】「災害対応に」の制御情報が与えられた場合には、遠隔地に設置されている記憶装置にデータを複写する。この場合、属性情報には記憶装置の設置場所を保持し、位置管理手段８０１は、異なる設置場所の２台以上の記憶装置にデータを記憶する。たとえば、ディスクアレイ装置２０１の設置場所が「東京」で、ディスクアレイ装置２０２の設置場所が「横浜」で、ディスクアレイ装置２０３の設置場所が「大阪」の場合、位置管理手段８０１は、設置場所の地理関係を把握し、「東京」のディスクアレイ装置２０１と「大阪」のディスクアレイ装置２０３の２箇所にデータを記録する。地震などの大災害時に、東京と横浜の距離では災害対応にならないことを位置管理手段８０１に設定しておく。

【００５９】ファイバチャネル５０の接続距離は最大１０ｋｍとなっており、実際に東京と横浜および大阪を接続するためには、ファイバチャネル５０をＡＴＭ（Asynchronous Transfer Mode）５１など他のネットワークに変換して接続する必要があるのはいうまでもない。

【００６０】「並列に」の制御情報が与えられた場合には、ディスクアレイ装置２０１と２０２の２箇所にデータを記録し、ディスクアレイ装置２０１に記録したデータをコンピュータ１０１専用とし、ディスクアレイ装置２０２に記録したデータをコンピュータ１０２専用とする。位置管理手段８０１は、読み出しを要求したコンピュータを判断して、対応する位置情報を報告する。

【００６１】「可搬に」の制御情報が与えられた場合には、媒体の取り外しが可能なライブラリ装置３０１にデータを記録する。このとき、制御情報にグループＩＤ（図２）を付加して位置管理手段８０１に与えることで、位置管理手段８０１は、同じグループＩＤをもつデータをひとつのメディアに集めるようにデータを記録する。「可搬に」の制御情報を扱うためには、変換表８１１にグループＩＤを保持する領域を設ける必要がある。

【００６２】また、これまで記載してきた実施形態では、データを新規に書き込むときに制御情報を与えることが必要であった。もちろん、ユーティリティを用いて途中で制御情報を変更することもできる。以下では、別の制御情報の与え方について説明する。

【００６３】図４は、ディレクトリ構造の例を示す図である。コンピュータシステムでは、ディレクトリ構造によるファイル管理を用いることが多い。これは、ファイルをフォルダに分類することにより、ユーザによるファイルの検索を容易にするためである。たとえば図４のような構成では、出発点になるルートフォルダ１には２つのサブフォルダ２と３があり、フォルダ３にはさらに２つのサブフォルダ４と５がある。データ「ＡＡＡ」が、このフォルダ４に収められているとすると、データ「ＡＡＡ」は、「／１／３／４／ＡＡＡ」となる。このようなディレクトリ構成では、フォルダに基本的な制御情報を与えておき、そのフォルダに格納されるファイルやサブフォルダには、そのフォルダの制御情報が継承されるように構成する。たとえば、フォルダ１には「制御情報なし」、フォルダ２には「高速に」、フォルダ３には「高信頼に」の制御情報を与えておけば、これらのフォルダに格納されるファイルについてはそれぞれの制御情報が継承される。また、フォルダ３にフォルダ４と５を作成すると、自動的にフォルダ３の制御情報である「高信頼に」がフォルダ４と５にも与えられる。継承させずに新規に制御情報を与えなおすこともできる。フォルダ４に「長期に」の制御情報を追加すると、フォルダ４に保持されるデータ「ＡＡＡ」には、デフォルトの制御情報として、「高信頼に」と「長期に」の制御情報が与えられることになる。

【００６４】また、フォルダ３の制御情報を変更すると、それ以下のフォルダとファイルすべての制御情報を一括して変更することも可能である。また、フォルダ４からフォルダ２にデータ「ＡＡＡ」を移動したときに、すでにデータ「ＡＡＡ」の持っている制御情報を維持することもできるし、フォルダ２の制御情報を与えなおすこともできる。本実施形態の場合、このような制御情報の継承に関するすべての設定は、位置管理手段８０１に対して設定できる。

【００６５】上記実施形態のシステムによれば、位置管理手段は、属性保持手段に保持された属性情報と、データに設定された制御情報によって、複数のコンピュータによって共有された複数の記憶装置の中から適切な記憶装置を選択し、データを保持することができる。位置管理手段は、データを記録する複数の位置情報をもつことができるので、データを複数の記憶装置に記録することができ、ユーザの設定したさまざまな制御情報にあわせたデータの記録が可能になる。またデータの位置は、すべて位置管理手段によって一元的に管理されるので、データの移動や複写を行うたびに位置管理手段が複数のコンピュータにその処理を報告する必要がない。

【００６６】次に、本発明の第２の実施形態（キャッシュありの場合）を説明する。

【００６７】図１は、第２の実施形態のシステム構成図を示す。第１の実施形態のシステムとの違いは、コンピュータ１０１と１０２にキャッシュディスク９０と情報受領手段９１を設けている点と、コンピュータ１０１，１０２、ディスクアレイ装置２０１，２０２，２０３、およびライブラリ装置３０１に稼働率計測手段８２を設

けている点である。コンピュータ１０１と１０２に設けたキャッシュディスク９０は、ファイルシステム１０とファイバチャネルスイッチ５０１から１つのWWNを持つハードディスク装置として認識されるように構成されている。

【００６８】次に、稼働率測定手段８２について説明する。コンピュータに設けた稼働率測定手段８２は、キャッシュディスク９０の処理性能を測定する。また、ディスクアレイ装置２０１，２０２，２０３に設けた稼働率測定手段８２は、ディスクアレイ装置の処理性能を測定する。本実施形態における処理性能とは、リードまたはライトにかかった平均処理時間である。ある記憶装置の平均処理時間が長くなると、その記憶装置にアクセスが集中していることが判る。ライブラリ装置３０１に設けた稼働率測定手段８２は、メディアごとのアクセス頻度を測定する。

【００６９】次に、位置管理手段８０１について説明する。第２の実施形態の位置管理手段８０１は、第１の実施形態と同様に図２の変換表８１１を用いる。領域管理表８１２は、ディスクアレイ装置２０１，２０２，２０３とライブラリ装置３０１のメディアに加えて、コンピュータ１０１と１０２に設けたキャッシュディスク９０の空き領域も管理する。

【００７０】次に、新規書き込み処理について説明する。図９と図１０は、記憶装置にデータを新規に書き込む処理手順を示すフローチャートである。コンピュータ１０１のファイルシステム１０は、イーサネット６０を介して、ファイルネーム「ＡＡＡ」というデータに対する新規領域割り当てをコンピュータ１５１の位置管理手段８０１に要求する（９００１）。位置管理手段８０１は、まず、変換表８１１の空いている一行を探し出し（９００２）、ライトビットを「１」にして書き込みの開始を宣言し（９００３）、ファイルネームを書き込む（９００４）。ここでは、変換表８１１の１番目の位置情報に書き込んだとする。コンピュータ１０１のキャッシュディスク９０から、データ「ＡＡＡ」を格納するのに必要な領域を領域管理表８１２より検索する（９００５）。ここでは、十分な空き領域があるとすると、割り当て可能位置を獲得することができる。位置管理手段８０１は、対応する領域管理表８１２のＬＢＡを割り当て済みに変更し（９００６）、位置情報をコンピュータ１０１に報告する（９００７）。

【００７１】コンピュータ１０１のファイルシステム１０は、空き領域の報告を受けると、指定された空き領域にデータを書き込むためのライトコマンドをコンピュータ１０１のキャッシュディスク９０に発行する（９００８）。この書き込みは同一コンピュータ内で処理されるため、きわめて高速である（９００９）。ファイルシステム１０は、キャッシュディスク９０から書き込み完了の報告（９０１０）を受け取ると、位置管理手段８０１に書き込みが完了したことを報告する（９０１１）。位置管理手段８０１は、書き込み完了の報告を受けとると、変換表８１１の１番目の位置情報に、コンピュータ１０１のキャッシュディスク９０のWWN、ＬＢＡ、およびデータ長を書き込み（９０１２）、有効ビットを「１」にセットする（９０１３）。

【００７２】次に、位置管理手段８０１は、コンピュータ１０１以外のコンピュータ上のキャッシュディスク９０とディスクアレイ装置２０１，２０２，２０３の稼働率測定手段８２から、それぞれ稼働率を獲得し（９０１４）、一番稼働率の低い記憶装置を選択し（９０１５）、空き領域を検索する。ここでは、ディスクアレイ装置２０１の稼動率が一番低いとすると、ディスクアレイ装置２０１に対しデータ「ＡＡＡ」を格納するのに必要な領域を検索する。データ「ＡＡＡ」を格納できる空き領域が見つかると（９０１６）、対応する領域管理表８１２のＬＢＡを割り当て済みに変更し（９０１７）、情報複製手段８９０にデータ「ＡＡＡ」の複製を要求する（１００００）。

【００７３】図１０に、情報複製手段８９０による複製処理の手順を示す。情報複製手段８９０は、すでにデータ「ＡＡＡ」が書き込まれているコンピュータ１０１のキャッシュディスク９０に対して情報受領手段９１を介してリードコマンドを発行し（１００００１）、データ「ＡＡＡ」を読み出し（１００００２，１００００３）、ディスクアレイ装置２０１にライトコマンドを発行し（１００００４）、データを書き込む（１００００５，１００００６）。情報複製手段８９０は、ライトコマンドを完了すると、位置管理手段８０１に複製が完了したことを報告する（１００００７）。

【００７４】再び図９に戻って、位置管理手段８０１は、複製完了の報告を受け取ると、変換表８１１の空いている行、ここでは２番目の位置情報に、ディスクアレイ装置２０１のWWN、ＬＢＡ、およびデータ長などの情報を書き込み（９０１９）、有効ビットを「１」にセットする（９０２０）。そして、ファイル履歴には、新規作成とその日時、複製日時を書き込む（９０２１）。最後に、ライトビットを「０」にして（９０２２）、書き込みを完了する（９０２３，９０２４，９０２５）。

【００７５】次に、読み出し処理について説明する。図１１と図１２は、記憶装置からデータを読み出す処理手順を示すフローチャートである。上記書き込み処理において、データ「ＡＡＡ」は、ホストコンピュータ１０１のキャッシュディスク９０と、ディスクアレイ装置２０１上にある。ここで、ホストコンピュータ１０２がデータ「ＡＡＡ」を読み出す場合（１１０００）、ホストコンピュータ１０２のファイルシステム１０が、イーサネット６０を介してファイルネーム「ＡＡＡ」のデータの読み出しを位置管理手段８０１に要求する（１１００１）。位置管理手段８０１は、変換表８１１からファイ

ルネーム「ＡＡＡ」に対応する行を選択し（１１００２）、ライトビットが立っていないことを確認し（１１００３）、リードカウントをインクリメントする（１１００４）。

【００７６】次に、位置管理手段８０１は、データ「ＡＡＡ」の格納されているホストコンピュータ１０１のキャッシュディスク９０とディスクアレイ装置２０１の稼働率計測手段８２から稼働率を獲得し（１１００６）、一番稼働率の低い記憶装置を選択する（１１００７）。ここでは、コンピュータ１０１のキャッシュディスク９０の稼働率が低かったとする。位置管理手段８０１は、領域管理表８１２のコンピュータ１０２のキャッシュディスク９０からデータ「ＡＡＡ」を格納するのに必要な領域を検索する（１１００８）。ここでは、十分な空き領域があるとすると、割り当て可能位置を獲得することができる。位置管理手段８０１は、対応する領域管理表８１２のＬＢＡを割り当て済みに変更し（１１００９）、情報複製手段８９０に対し、コンピュータ１０１のキャッシュディスク９０からコンピュータ１０２のキャッシュディスク９０にデータ「ＡＡＡ」をコピーするように要求する（１２０００）。

【００７７】図１２に、情報複製手段８９０による複製処理の手順を示す。情報複製手段８９０は、すでにデータ「ＡＡＡ」が書き込まれているコンピュータ１０１のキャッシュディスク９０に対してリードコマンドを発行し（１２００１）、データを読み出し（１２００２，１２００３）、コンピュータ１０２のキャッシュディスク９０にライトコマンドを発行し（１２００４）、データを書き込む（１２００５，１２００６）。情報複製手段８９０は、ライトコマンドを完了すると、位置管理手段８０１に複製が完了したことを報告する（１２００７）。

【００７８】再び図１１に戻って、位置管理手段８０１は、情報複製手段８９０から複製の完了を受け取ると、変換表８１１の空いている行、ここでは３番目の位置情報にホストコンピュータ１０２のキャッシュディスク９０のWWN、ＬＢＡ、およびデータ長などの情報を書き込み（１１０１１）、有効ビットを「１」にセットする（１１０１２）。また、位置管理手段８０１は、ファイル履歴に複製日時を書き込む（１１０１３）。そしてデータ「ＡＡＡ」の入っているホストコンピュータ１０２のキャッシュディスク９０の位置情報をコンピュータ１０２に報告する（１１０１４）。

【００７９】コンピュータ１０２のファイルシステム１０は、位置情報（WWN、ＬＢＡ、データ長）の報告を受けると、コンピュータ１０２のキャッシュディスク９０にリードコマンドを発行し（１１０１５）、データを読み出す（１１０１６）。コンピュータ１０２のファイルシステム１０は、キャッシュディスク９０から読み出し完了の報告（１１０１７）を受け取ると、コンピュータ１５１の位置管理手段８０１に読み出しが完了したことを報告する（１１０１８）。位置管理手段８０１は、読み出し完了の報告を受け取ると、ファイル履歴に最終アクセス日時を書き込み（１１０１９）、リードカウントをデクリメントし（１１０２０）、読み出しが完了する（１１０２１，１１０２２，１１０２３）。

【００８０】次に、データの更新処理について説明する。図１３と図１４は、データを上書きし更新する処理手順を示すフローチャートである。上記書き込み処理と読み出し処理において、データ「ＡＡＡ」は、ホストコンピュータ１０１と１０２のキャッシュディスク９０と、ディスクアレイ装置２０１上の３箇所にある。ここで、ホストコンピュータ１０２がデータ「ＡＡＡ」を更新する場合（１３０００）、ホストコンピュータ１０２のファイルシステム１０は、イーサネット６０を介して、ファイルネーム「ＡＡＡ」のデータの更新を位置管理手段８０１に要求する（１３００１）。位置管理手段８０１は、変換表８１１からファイルネーム「ＡＡＡ」に対応する行を選択し（１３００２）、ライトビットが「０」でリードカウントが「０」であることを確認する（１３００３，１３００４）。ライトビットが「０」でない場合、またはリードカウントが「０」でない場合は、更新失敗となる（１３０９０，１３０９１，１３０９８，１３０９９）。

【００８１】次に、位置管理手段８０１は、ライトビットを「１」にして更新の開始を宣言する（１３００５）。もしデータ「ＡＡＡ」のデータ長が大きくなっていた場合には、領域管理表８１２を用いて再度空き領域を検索する必要があるが（１３００６，１３００７）、ここではデータ長が不変とする。位置管理手段８０１は、コンピュータ１０２のファイルシステム１０に、変換表８１１の３番目の位置情報（コンピュータ１０２のキャッシュディスク９０内のデータ「ＡＡＡ」を示す）を報告する（１３００８）。

【００８２】コンピュータ１０２のファイルシステム１０は、位置情報の報告を受けると、指定された位置にデータを書き込むためのライトコマンドをコンピュータ１０２のキャッシュディスク９０に発行する（１３００９）。コンピュータ１０２のキャッシュディスク９０は、データを書き込み（１３０１０）、書き込み完了を報告する（１３０１１）。ファイルシステム１０は、キャッシュディスク９０から書き込み完了の報告を受け取ると、コンピュータ１５１の位置管理手段８０１に書き込みが完了したことを報告する（１３０１２）。

【００８３】位置管理手段８０１は、書き込み完了の報告を受けとると、変換表８１１の１番目（コンピュータ１０１のキャッシュディスク９０内のデータ「ＡＡＡ」を示す）と２番目（ディスクアレイ２０１内のデータ「ＡＡＡ」を示す）の位置情報に、更新したデータ「ＡＡＡ」を複製するように、情報複製手段８９０に要求す

る（１４０００）。

【００８４】図１４に、情報複製手段８９０による複製処理の手順を示す。情報複製手段８９０は、すでにデータ「ＡＡＡ」が書き込まれているコンピュータ１０２のキャッシュディスク９０に対してリードコマンドを発行し（１４００１）、データを読み出し（１４００２，１４００３）、コンピュータ１０１のキャッシュディスク９０とディスクアレイ装置２０１にライトコマンドを同時に発行し（１４００４）、データを書き込む（１４００５，１４００６，１４００５’，１４００６’）。情報複製手段８９０は、二つのライトコマンドを完了すると、位置管理手段８０１に複製が完了したことを報告する（１４００７）。本実施形態における情報複製手段８９０は、ファイバチャネルスイッチ５０１と連携し、１つのデータを複数の記憶装置に同時に複製することができる。もし、情報複製手段８９０が、１つの記憶装置にしかデータを複製できない場合には２回に分けて処理する。この場合、処理時間が長くなることが予想されるので、コピー先が多くなればなるほど不利になる。本発明の実施には、複数の記憶装置にデータを複製可能な情報複製手段８９０を用いることが望ましい。

【００８５】再び図１３に戻って、位置管理手段８０１は、情報複製手段８９０から複製完了の報告を受け取ると、ファイル履歴に最終更新日時を書き込み（１３０１４）、ライトビットを「０」にして（１３０１５）、書き込みを完了する（１３０１６，１３０１７，１３０１８）。

【００８６】上記第２の実施の形態のシステムによれば、ホストコンピュータにディスクキャッシュを設けることで、ホストコンピュータ上に書き込みして間もないデータを保持できるので、高速にアクセスできるようになる。位置管理手段は、各記憶装置に設けた稼動率測定手段からアクセス頻度の少ない記憶装置を選択してデータを２箇所以上に保存するので、ホストコンピュータが停止するようなことがあってもデータを失うことはない。

【００８７】次に、本発明の第３の実施形態（ストレージプロバイダにおける保存データの課金方式）について説明する。

【００８８】図１は、第３の実施形態のシステム構成図を示す。第３の実施形態では、イーサネット６１に複数のクライアント７０を接続している。コンピュータ１０１と１０２は、これらクライアント７０からのデータの書き込みおよび読み出し要求を受け付けるファイルサーバとして機能する。本実施形態ではクライアント７０との接続にイーサネット６１を用いて説明を行うが、本発明はこの接続手段によらず実現可能である。

【００８９】本実施形態ではコンピュータ１０１と１０２はストレージサービスプロバイダ側に設置されているとして説明を行うが、もちろんコンピュータ１０１や１０２を各事業所や支店に設置することもできる。もし、コンピュータ１０２とファイバチャネルスイッチ５０１との距離が１０ｋｍ以上離れる場合には、図１に記載のように、エクステンダ５２を介し、ＡＴＭ（Asynchronous Transfer Mode）５１を介して接続する。もちろんＡＴＭ５１に代えて他の接続手段を用いても問題ない。

【００９０】次に、運用形態について説明する。クライアント７０はたとえば各事業所や支店などに用意された端末であり、ユーザは、これらの端末から、保存を希望するデータをイーサネット６１を介してコンピュータ１０１または１０２に送りつける。この際、データをどのように保存するかを指定する（データ毎に指定する保存の仕方をデータポリシーと称す）。第１の実施形態に記載した制御情報がこれにあたる。ユーザは、アクセス頻度の高いデータはより高速な記憶装置に、アクセス頻度の低いデータはアクセスが遅くても低コストな記憶装置に、また、大切なデータは、コストがかかっても災害でデータを失うことのないように遠隔地の記憶装置にも複製を保持することを希望する。よって、データの保持される記憶装置の属性（性能や信頼性）が、データの保持コストに反映されることが望ましい。

【００９１】次に、課金について説明する。本実施形態のシステムでは、記憶装置毎の属性保持手段８０に保持した単位容量あたりの保持コストと、位置管理情報の変換表８１１に保持したデータ長、ファイル履歴、および位置情報をもとに、保持データに対して課金する。たとえば、データ「ＡＡＡ」がディスクアレイ装置２０１と２０２に１０日間保持され、次にディスクアレイ装置２０３に移動されて１０日間保存され、次にライブラリ装置３０１に移動されて１０日間保存された場合、この３０日間のデータ保持コストは、

データ保持コスト＝｛（ＲＡＩＤ０の保持コスト１０）×１０日×２＋（ＲＡＩＤ５の保持コスト１５）×１０日＋（ライブラリの保持コスト１）×１０日｝×データ長

で求めることができる。ここで、ＲＡＩＤ０の保持コストが２倍されているのは、２台のディスクアレイにデータが二重に保持されていたことを反映するためである。位置管理手段８０１は、一ヶ月ごとにすべてのデータのコストを算出し、ユーザに請求する。コスト算出後、位置管理手段８０１は、ファイル履歴のうち、データの移動と複製に関する情報は消去する。上記コストの算出に必要な情報は、第１の実施形態および第２の実施形態に含まれている。

【００９２】また、別の例として、データ「ＢＢＢ」が、２枚のＤＶＤ－ＲＡＭメディアに３０日間保持され、一度参照したために、ディスクアレイ装置２０１に３日間キャッシュされた場合、この３０日間のデータ保持コストは、

データ保持コスト＝｛（ＲＡＩＤ０の保持コスト１０）

×３日＋（ライブラリの保持コスト１）×３０日×２）
×データ長
で求めることができる。

【００９３】ユーザ毎のデータコストを算出するには、WWNを用いる。たとえば、ユーザＡがコンピュータ１０１を介してデータを保存し、ユーザＢがホストコンピュータ１０２を介してデータを保持した場合、ホストコンピュータ１０１と１０２は異なるWWNを有するので、目的のホストコンピュータのWWNで変換表８１１を検索すれば、そのホストコンピュータを使用するユーザの総データ保持コスト算出することができる。

【００９４】次に、グローバルポリシーの適用について説明する。上述したように本実施形態のシステムでは、データ単位でのコストを容易に算出することが出来る。そこで、ユーザからたとえば一ヶ月あたりのデータ保持コストが指定されていた場合には、そのコストを満足できるように、データを移動させる。当然ながらデータ量が多くなると性能が犠牲になる。

【００９５】このような指定を、データポリシーに対して、グローバルポリシーと称す。コスト以外にも性能最優先のグローバルポリシーなどが考えられる。また、データポリシーとグローバルポリシーのどちらを優先するかも指定可能である。もし、データポリシーが優先されていて、ユーザから指定されたデータ保持コスト（グローバルポリシー）を満たすことが出来ない場合には、その旨をユーザに報告する。　通常、グローバルポリシーの設定は、ユーザ側のクライアント７０からホストコンピュータ１０１または１０２を介してコンピュータ１５１に通知するが、ユーザとストレージサービスプロバイダが書面により交換したグローバルポリシーの設定基準に基づき、ストレージサービスプロバイダ側の管理者がコンピュータ１５１に設定することもできる。

【００９６】上記第３の実施形態のシステムによれば、ユーザから保持要求のあったデータを、ユーザの指定した制御情報と、属性保持手段、位置管理手段、および情報複製手段とにより、よりきめ細かく位置管理することができる。これにより、データのアクティビティや信頼性に対し、データ保持にかかるコストを関連付けることができ、ユーザは、高いアクティビティや高い信頼性を必要とするデータ対しては高い保持コストを、低いアクティビティや低い信頼性のデータには安い保持コストを割り当てることができる。一般的に、高いアクティビティを必要とするデータは少ないと言えるが、本実施形態のシステムでは、上述したようにデータのアクティビティや信頼性に応じたコストを関連付けることができるので、全体としてユーザはデータを保持しつづけるコストを低減できる。

【００９７】

【発明の効果】以上説明したように、本発明によれば、位置管理手段が、ストレージの稼働率、データの利用状況、制御情報、および課金情報などに基づいてデータを移動および複製することで、ユーザの求める性能や信頼性、保存性、およびコストに対応したデータの保存が可能になる。また、複数のコンピュータ、複数のストレージのデータの位置管理を位置管理手段で集中して行うことで、データを移動したことを各コンピュータに通知する必要がない。また、情報複製手段により、同時に複数のデータの複製を可能とすることで、コンピュータの台数が多い場合にも、複製にかかる時間を低減することができる。

【図面の簡単な説明】

【図１】本発明の実施形態を示すシステム構成図である。

【図２】変換表を示す図である。

【図３】領域管理表を示す図である。

【図４】ディレクトリ構成を示す図である。

【図５】第１の実施形態における新規書き込み処理を示すフローチャート図である。

【図６】第１の実施形態における読み出し処理を示すフローチャート図である。

【図７】第１の実施形態における位置管理手段の監視と移動処理を示すフローチャート図である。

【図８】複製処理を示すフローチャート図である。

【図９】第２の実施形態における新規書き込み処理を示すフローチャート図である。

【図１０】複製処理を示すフローチャート図である。

【図１１】第２の実施形態における読み出し処理を示すフローチャート図である。

【図１２】複製処理を示すフローチャート図である。

【図１３】第２の実施形態におけるデータの更新処理を示すフローチャート図である。

【図１４】複製処理を示すフローチャート図である。

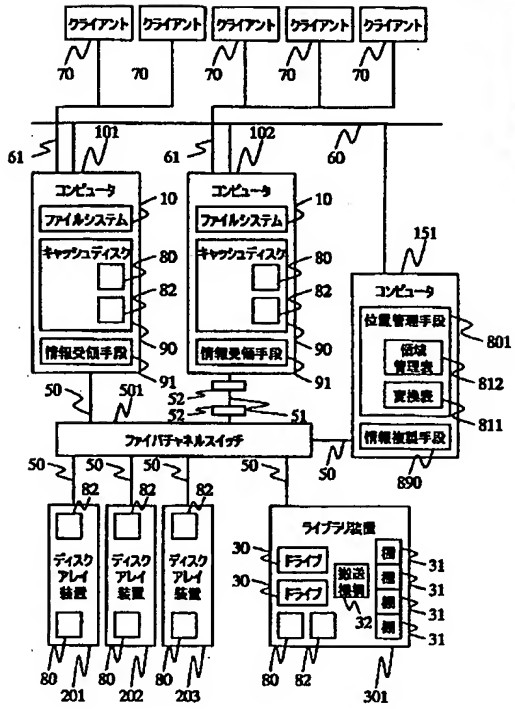
【図１５】第１の実施形態においてライブラリ装置に格納されたデータを読み出す処理を示すフローチャート図である。

【図１６】複製処理を示すフローチャート図である。

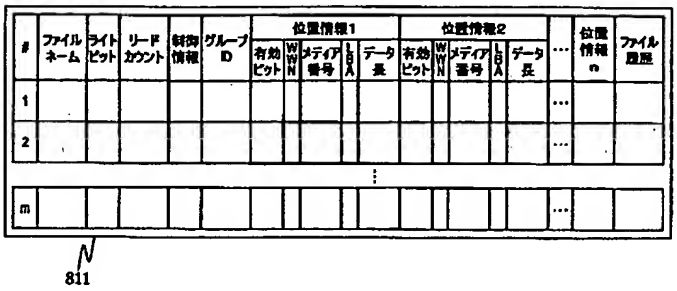
【符号の説明】

１０…ファイルシステム、３０…ドライブ、３１…棚、３２…搬送機構、５０…ファイバチャネル、５１…ＡＴＭ、５２…エクステンダ、６０…イーサネット、６１…イーサネット、７０…クライアント、８０…属性保持手段、８２…稼働率測定手段、９０…キャッシュディスク、９１…情報受領手段、１０１…コンピュータ、１０２…コンピュータ、１５１…コンピュータ、２０１…ディスクアレイ装置、２０２…ディスクアレイ装置、２０３…ディスクアレイ装置、３０１…ライブラリ装置、５０１…ファイバチャネルスイッチ、８０１…位置管理手段、８１１…変換表、８１２…領域管理手段、８９０…情報複製手段。

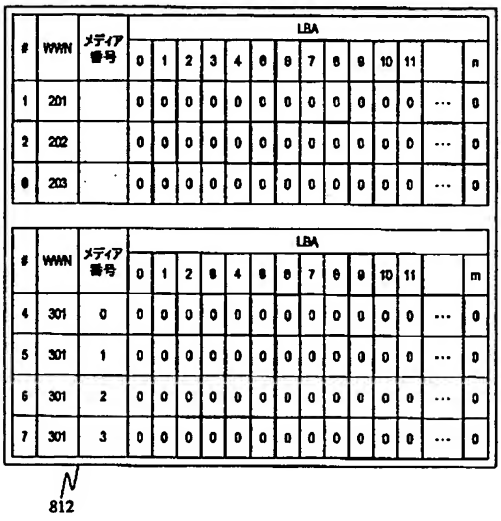
【図１】


クライアント
70
101
102
61
60
コンピュータ
ファイルシステム
キャッシュディスク
情報受領手段
情報受領手段
位置管理手段
領域管理表
変換表
情報複製手段
10
80
82
90
91
151
801
812
811
890
50
501
52
51
ファイバチャネルスイッチ
ディスクアレイ装置
ライブラリ装置
Fドライブ
搬送機構
テープ
30
31
32
201
202
203
301

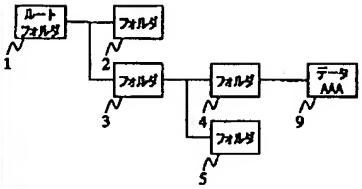
【図２】


#
ファイルネーム
ライトビット
リードカウント
制御情報
グループID
位置情報1
位置情報2
有効ビット
WWN
メディア番号
LBA
データ長
位置情報n
ファイル履歴
1
2
m
811

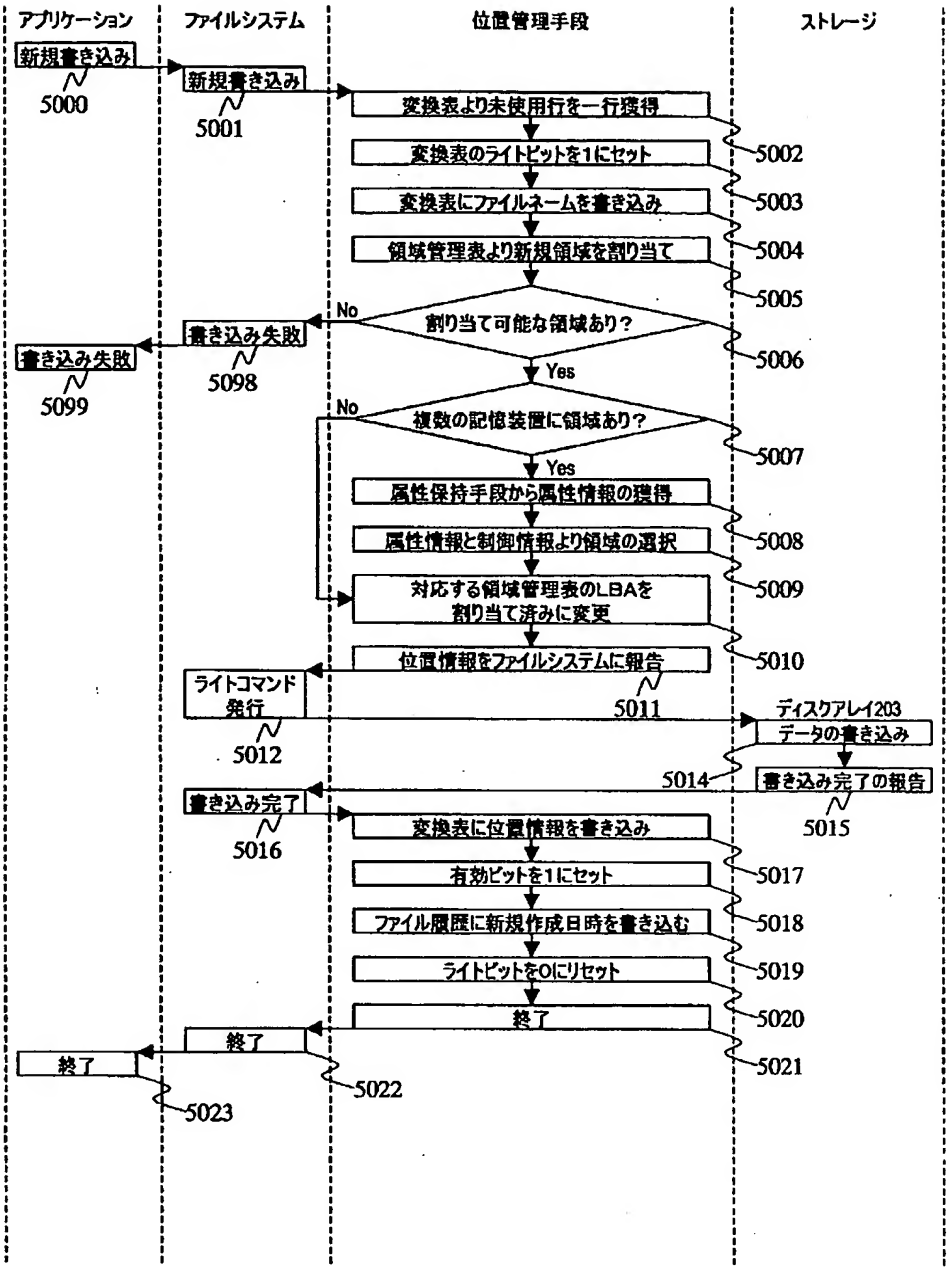
【図３】


#
WWN
メディア番号
LBA
0 1 2 3 4 5 6 7 8 9 10 11 … n
1 201 0 0 0 0 0 0 0 0 0 0 0 0 … 0
2 202 0 0 0 0 0 0 0 0 0 0 0 0 … 0
3 203 0 0 0 0 0 0 0 0 0 0 0 0 … 0
4 301 0 0 0 0 0 0 0 0 0 0 0 0 … 0
5 301 1 0 0 0 0 0 0 0 0 0 0 0 … 0
6 301 2 0 0 0 0 0 0 0 0 0 0 0 … 0
7 301 3 0 0 0 0 0 0 0 0 0 0 0 … 0
m
812

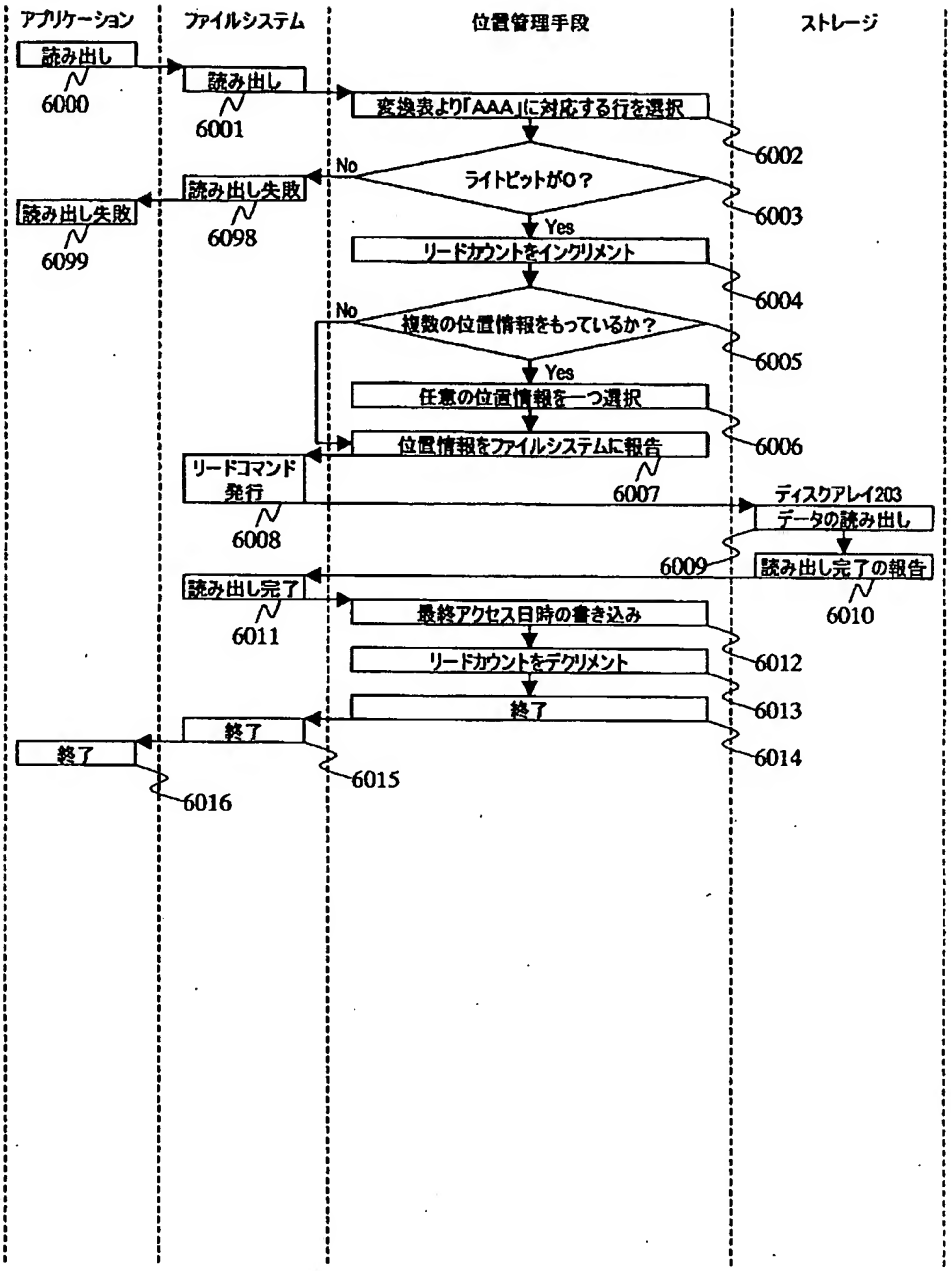
【図４】


ルートフォルダ
フォルダ
データAAA
1
2
3
4
5
9

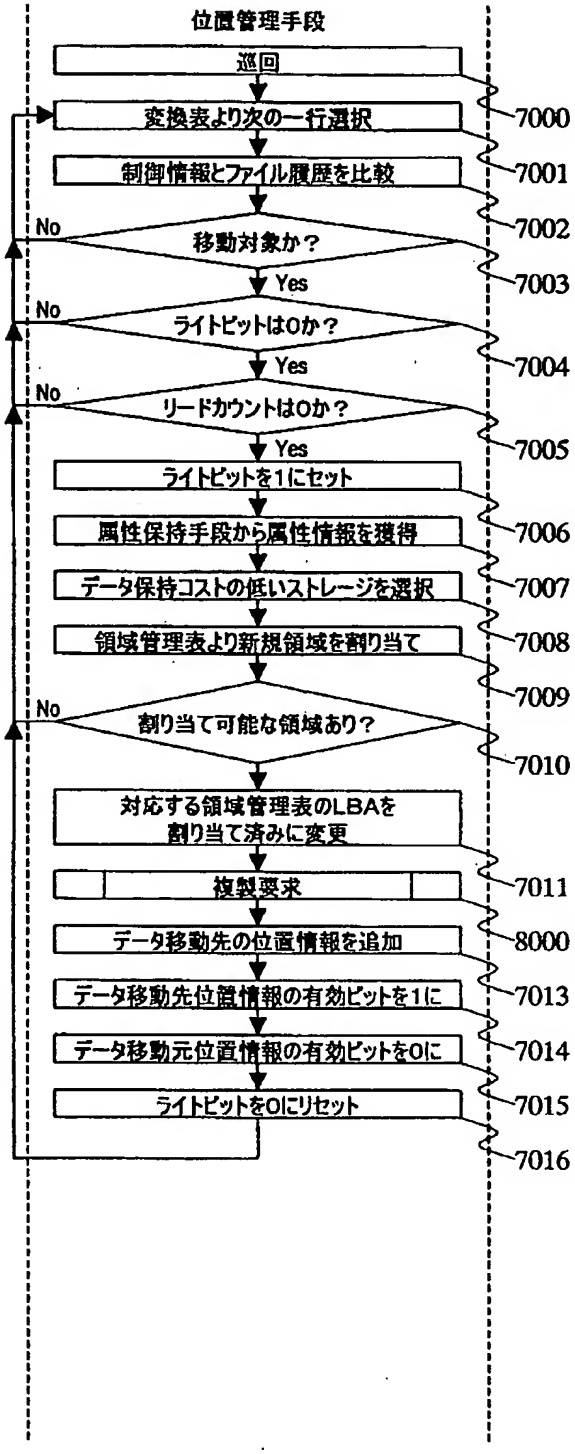
【図５】


アプリケーション
ファイルシステム
位置管理手段
ストレージ
新規書き込み
5000
新規書き込み
5001
変換表より未使用行を一行獲得
5002
変換表のライトビットを1にセット
5003
変換表にファイルネームを書き込み
5004
領域管理表より新規領域を割り当て
5005
No
割り当て可能な領域あり？
5006
Yes
書き込み失敗
5098
書き込み失敗
5099
No
複数の記憶装置に領域あり？
5007
Yes
属性保持手段から属性情報の獲得
5008
属性情報と制御情報より領域の選択
5009
対応する領域管理表のLBAを
割り当て済みに変更
5010
位置情報をファイルシステムに報告
5011
ライトコマンド
発行
5012
ディスクアレイ203
データの書き込み
5014
書き込み完了の報告
5015
書き込み完了
5016
変換表に位置情報を書き込み
5017
有効ビットを1にセット
5018
ファイル履歴に新規作成日時を書き込む
5019
ライトビットを0にリセット
5020
終了
5021
終了
5022
終了
5023

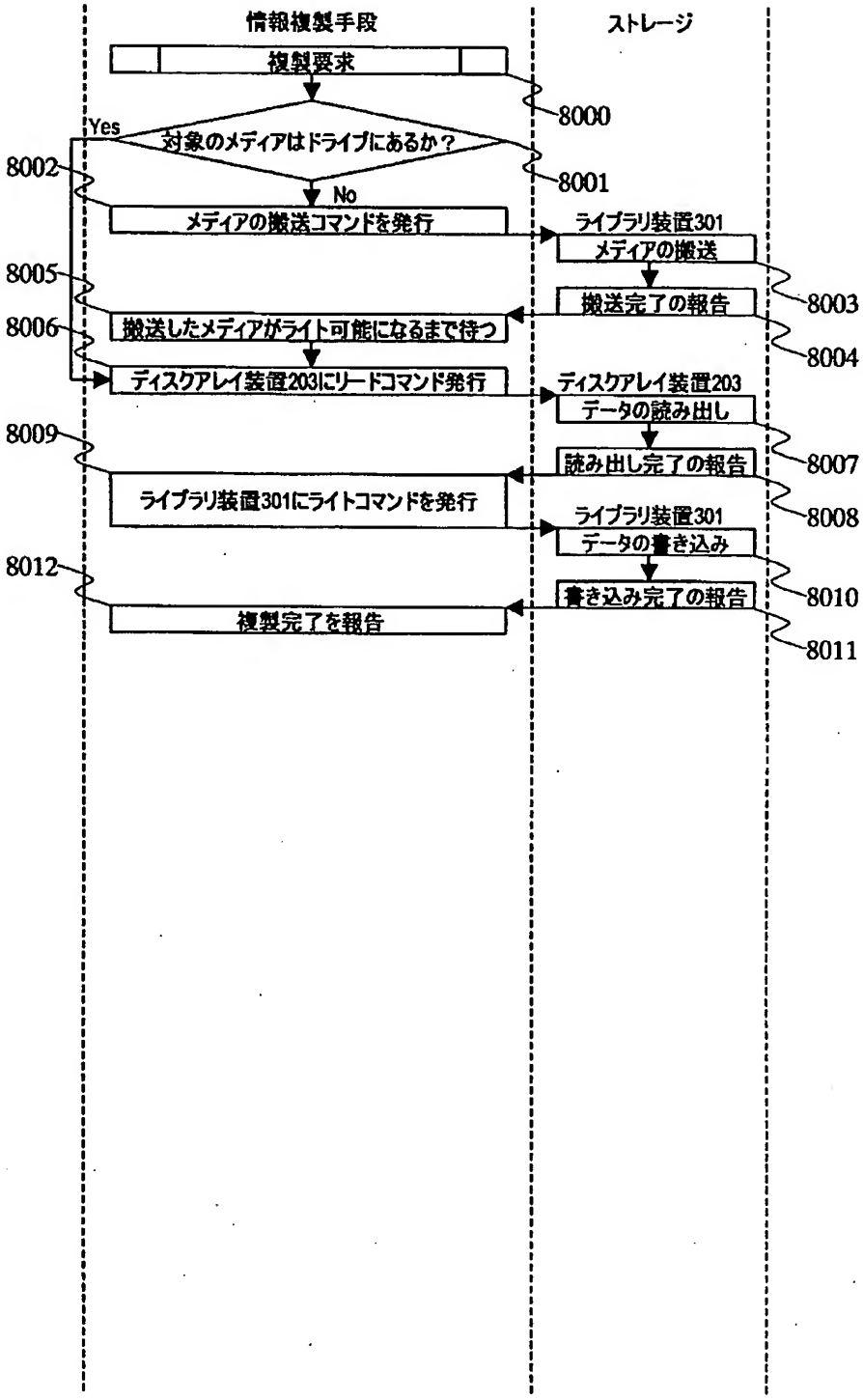
【図６】


アプリケーション
ファイルシステム
位置管理手段
ストレージ
読み出し
6000
読み出し
6001
変換表より「AAA」に対応する行を選択
6002
No
ライトビットが0？
6003
Yes
読み出し失敗
6098
読み出し失敗
6099
リードカウントをインクリメント
6004
No
複数の位置情報をもっているか？
6005
Yes
任意の位置情報を一つ選択
6006
位置情報をファイルシステムに報告
6007
リードコマンド発行
6008
ディスクアレイ203
データの読み出し
6009
読み出し完了の報告
6010
読み出し完了
6011
最終アクセス日時の書き込み
6012
リードカウントをデクリメント
6013
終了
6014
終了
6015
終了
6016

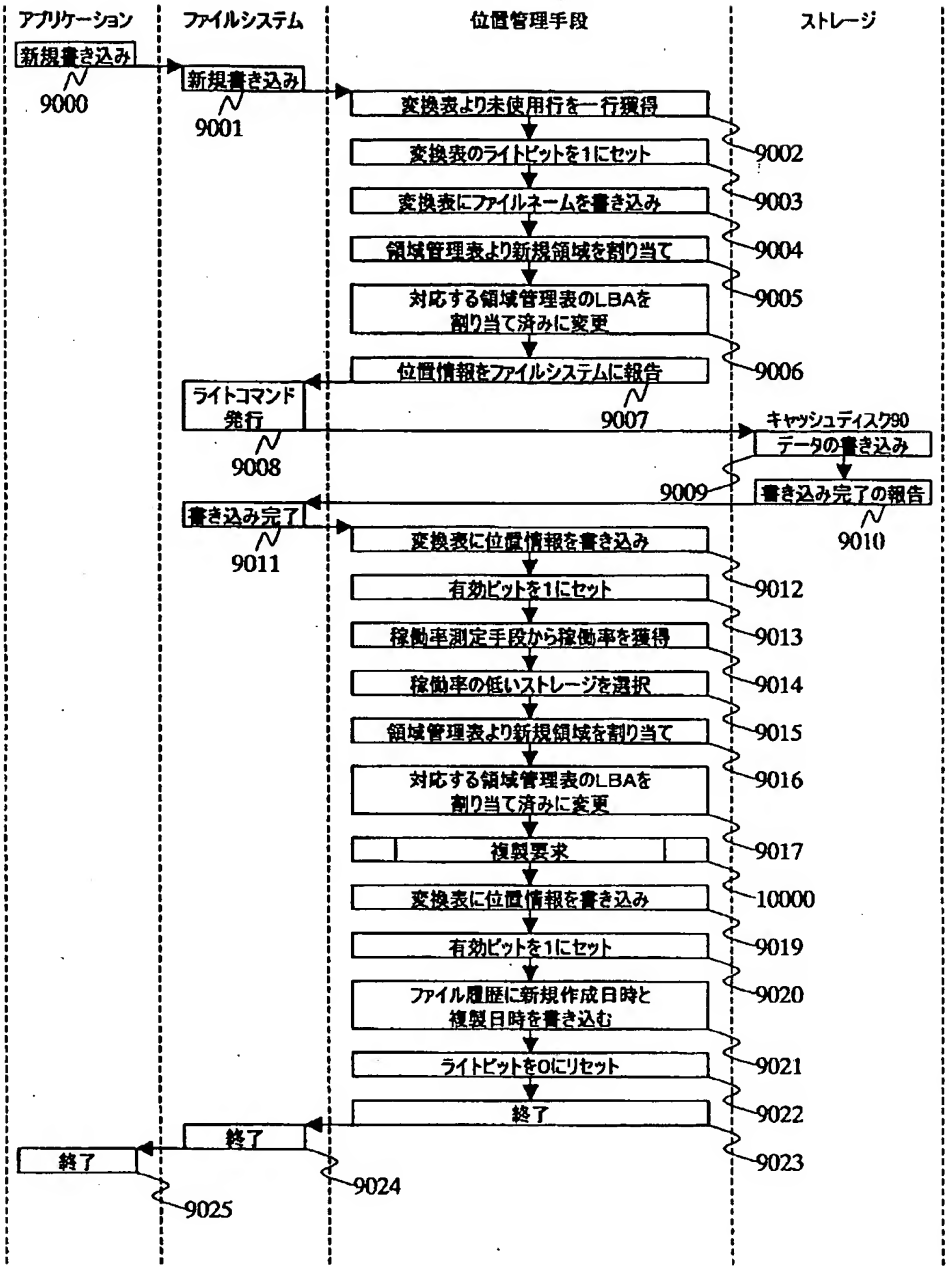
【図７】


位置管理手段
巡回
変換表より次の一行選択
制御情報とファイル履歴を比較
No
移動対象か？
Yes
No
ライトビットは0か？
Yes
No
リードカウントは0か？
Yes
ライトビットを1にセット
属性保持手段から属性情報を獲得
データ保持コストの低いストレージを選択
領域管理表より新規領域を割り当て
No
割り当て可能な領域あり？
対応する領域管理表のLBAを
割り当て済みに変更
複製要求
データ移動先の位置情報を追加
データ移動先位置情報の有効ビットを1に
データ移動元位置情報の有効ビットを0に
ライトビットを0にリセット
7000
7001
7002
7003
7004
7005
7006
7007
7008
7009
7010
7011
8000
7013
7014
7015
7016

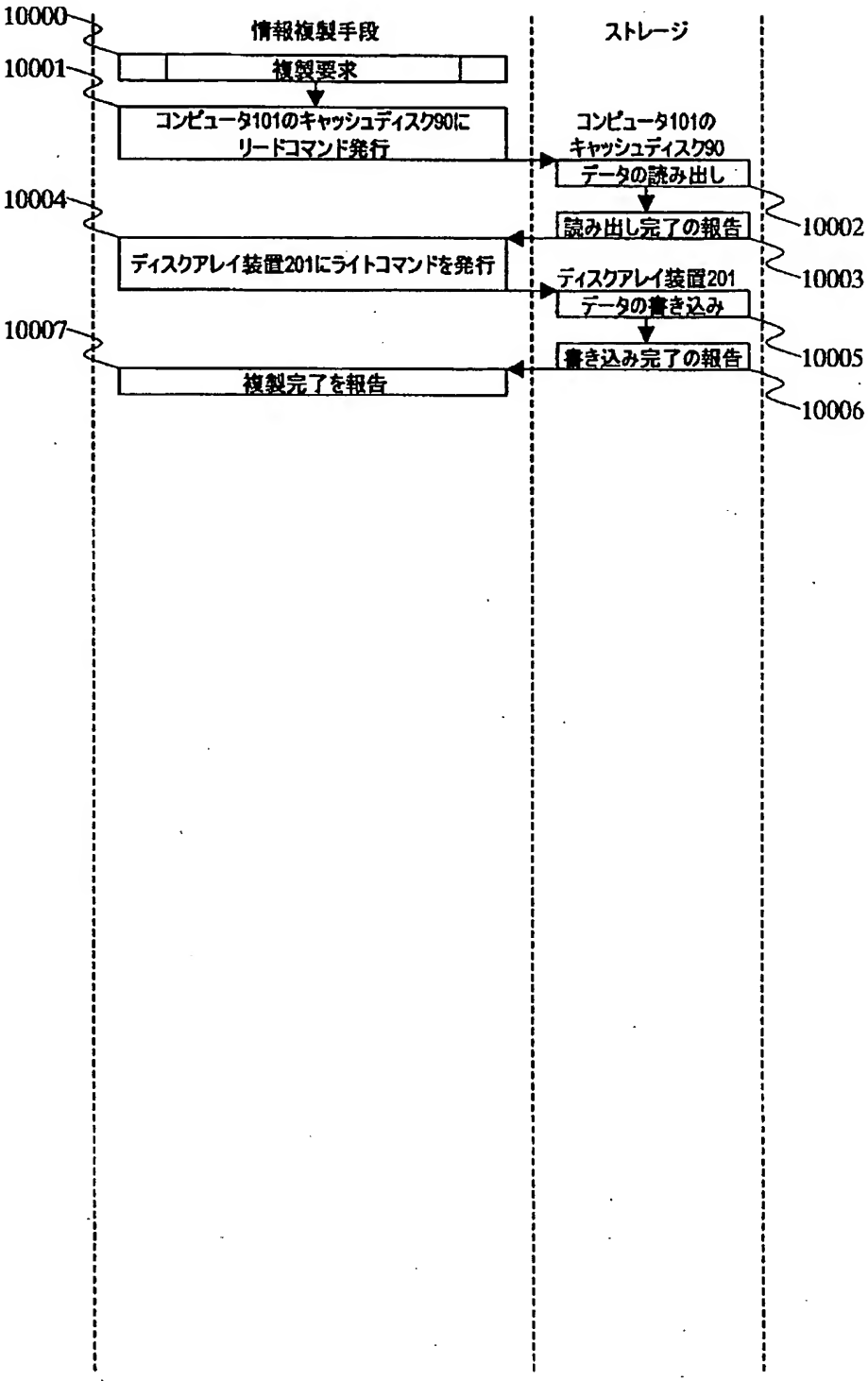
【図８】


情報複製手段
ストレージ
複製要求
8000
対象のメディアはドライブにあるか？
Yes
No
8001
8002
メディアの搬送コマンドを発行
ライブラリ装置301
メディアの搬送
8003
搬送完了の報告
8004
8005
搬送したメディアがライト可能になるまで待つ
8006
ディスクアレイ装置203にリードコマンド発行
ディスクアレイ装置203
データの読み出し
8007
読み出し完了の報告
8008
8009
ライブラリ装置301にライトコマンドを発行
ライブラリ装置301
データの書き込み
8010
書き込み完了の報告
8011
8012
複製完了を報告

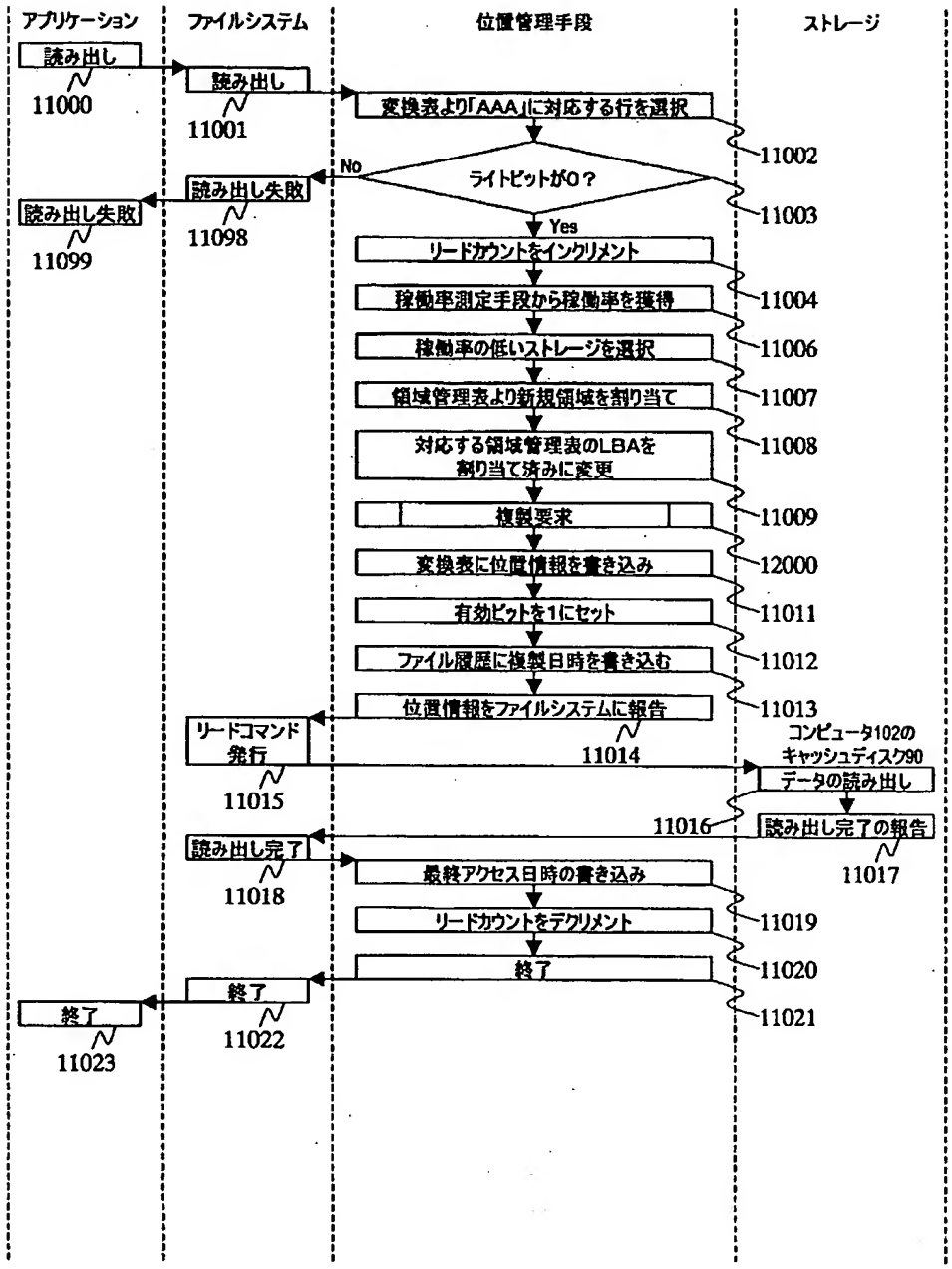
【図９】


アプリケーション
ファイルシステム
位置管理手段
ストレージ
新規書き込み
9000
新規書き込み
9001
変換表より未使用行を一行獲得
変換表のライトビットを1にセット
9002
変換表にファイルネームを書き込み
9003
領域管理表より新規領域を割り当て
9004
対応する領域管理表のLBAを
割り当て済みに変更
9005
位置情報をファイルシステムに報告
9006
ライトコマンド
発行
9007
キャッシュディスク90
データの書き込み
9008
9009
書き込み完了の報告
9010
書き込み完了
9011
変換表に位置情報を書き込み
9012
有効ビットを1にセット
9013
稼働率測定手段から稼働率を獲得
9014
稼働率の低いストレージを選択
9015
領域管理表より新規領域を割り当て
9016
対応する領域管理表のLBAを
割り当て済みに変更
9017
複製要求
10000
変換表に位置情報を書き込み
9019
有効ビットを1にセット
9020
ファイル履歴に新規作成日時と
複製日時を書き込む
9021
ライトビットを0にリセット
9022
終了
9023
終了
9024
終了
9025

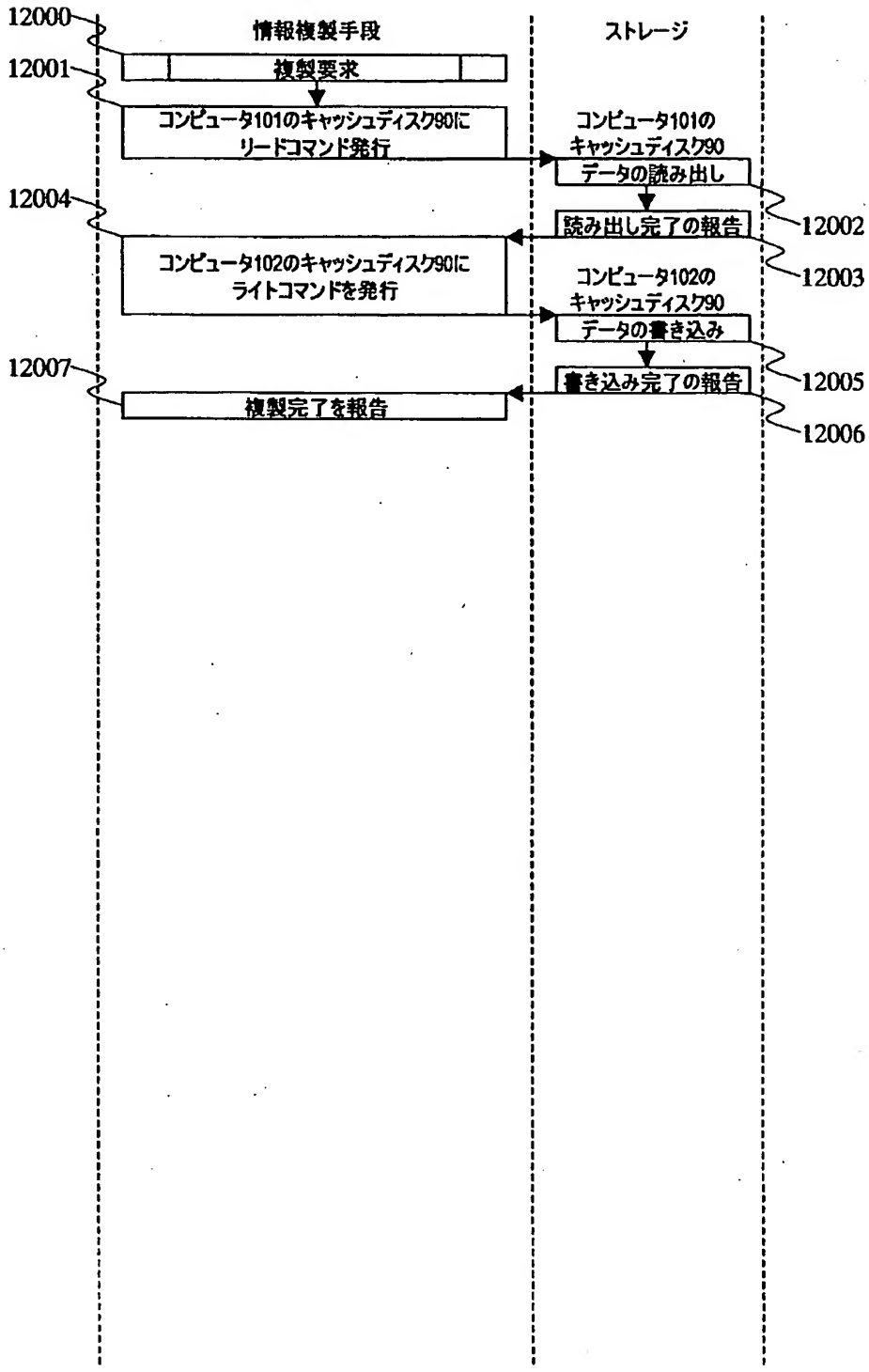
【図１０】


10000
情報複製手段
ストレージ
10001
複製要求
コンピュータ101のキャッシュディスク90に
リードコマンド発行
コンピュータ101の
キャッシュディスク90
データの読み出し
10002
読み出し完了の報告
10003
10004
ディスクアレイ装置201にライトコマンドを発行
ディスクアレイ装置201
データの書き込み
10005
書き込み完了の報告
10006
10007
複製完了を報告

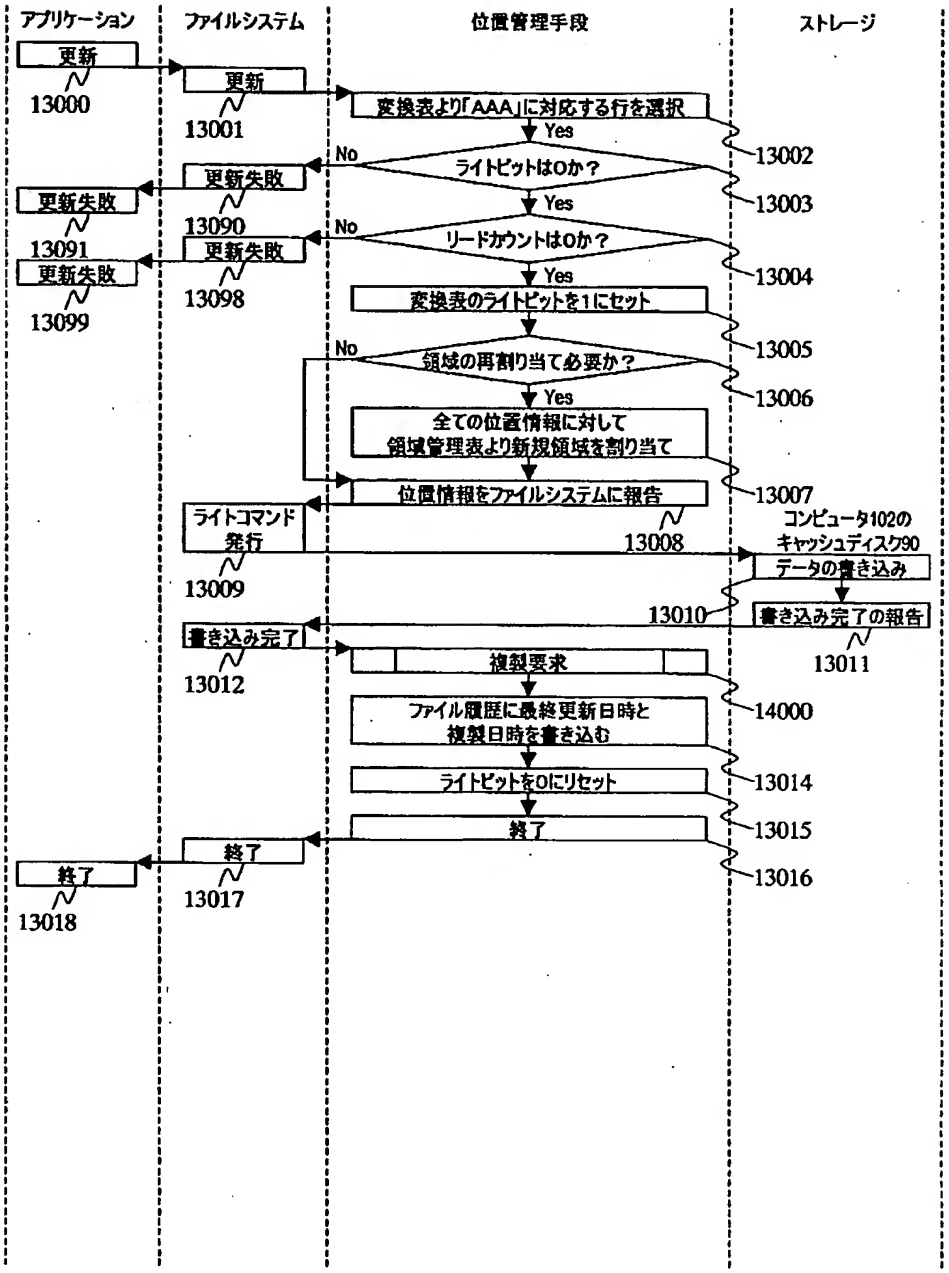
【図１１】


アプリケーション
ファイルシステム
位置管理手段
ストレージ
読み出し
11000
読み出し
11001
変換表より「AAA」に対応する行を選択
11002
No
ライトビットが0？
11003
Yes
読み出し失敗
11098
読み出し失敗
11099
リードカウントをインクリメント
11004
稼働率測定手段から稼働率を獲得
11006
稼働率の低いストレージを選択
11007
領域管理表より新規領域を割り当て
11008
対応する領域管理表のLBAを
割り当て済みに変更
11009
複製要求
12000
変換表に位置情報を書き込み
11011
有効ビットを1にセット
11012
ファイル履歴に複製日時を書き込む
11013
位置情報をファイルシステムに報告
11014
リードコマンド
発行
11015
コンピュータ102の
キャッシュディスク90
データの読み出し
11016
読み出し完了の報告
11017
読み出し完了
11018
最終アクセス日時の書き込み
11019
リードカウントをデクリメント
11020
終了
11021
終了
11022
終了
11023

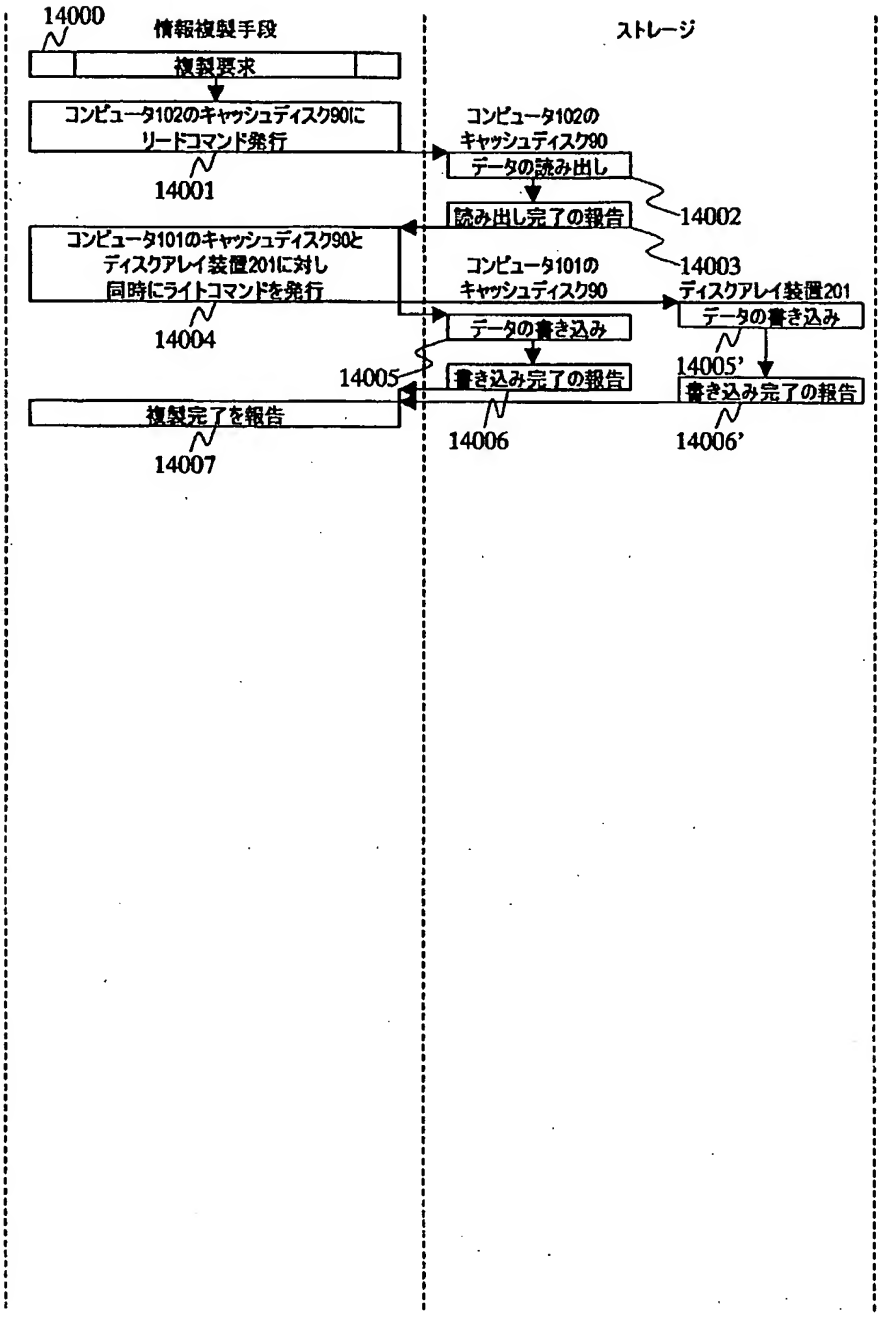
【図１２】


12000
情報複製手段
ストレージ
12001
複製要求
コンピュータ101のキャッシュディスク90に
リードコマンド発行
コンピュータ101の
キャッシュディスク90
データの読み出し
12002
読み出し完了の報告
12003
12004
コンピュータ102のキャッシュディスク90に
ライトコマンドを発行
コンピュータ102の
キャッシュディスク90
データの書き込み
12005
書き込み完了の報告
12006
12007
複製完了を報告

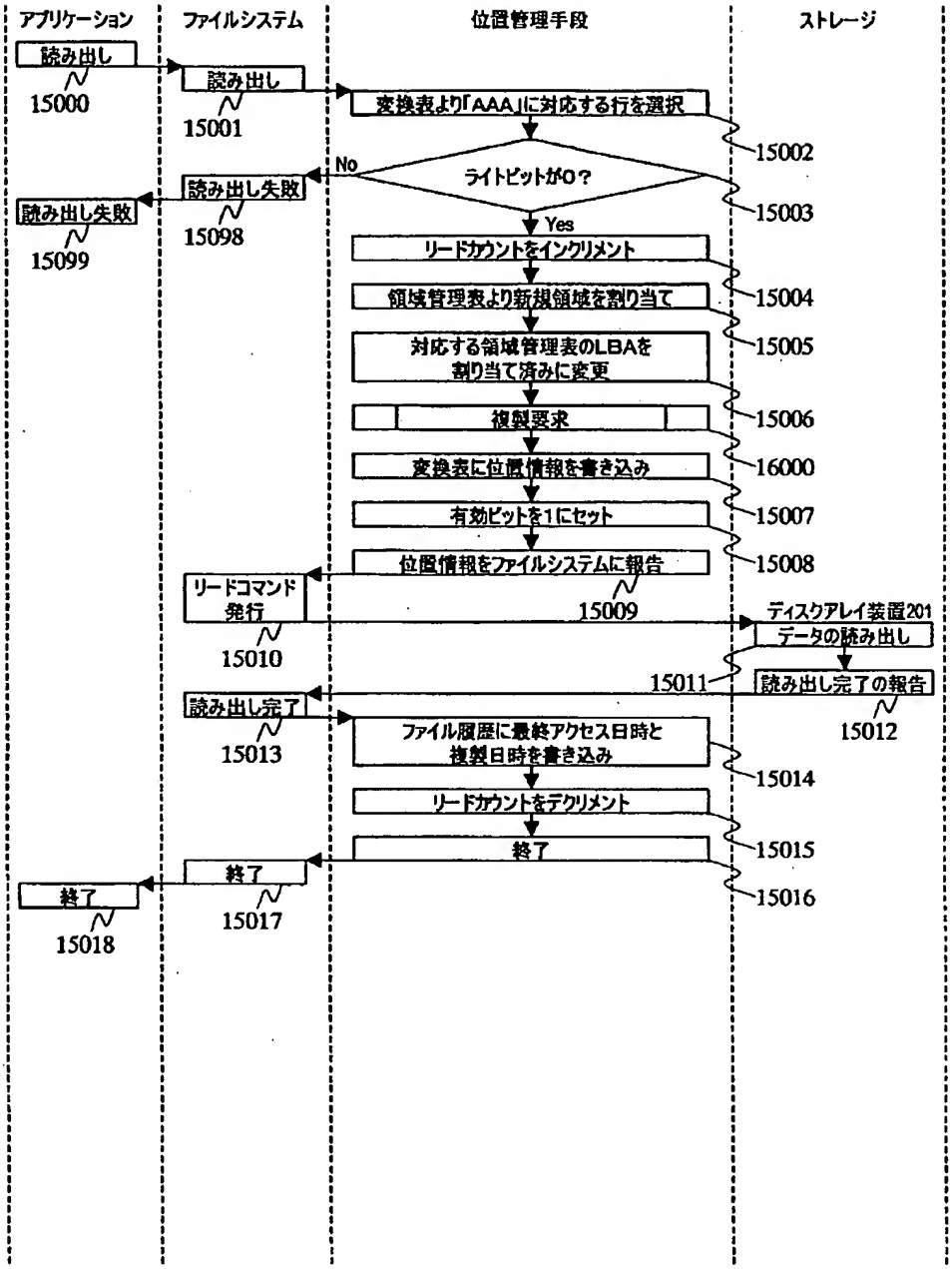
【図１３】


アプリケーション
ファイルシステム
位置管理手段
ストレージ
更新
13000
更新
13001
変換表より「AAA」に対応する行を選択
13002
Yes
No
ライトビットは0か？
13003
Yes
No
リードカウントは0か？
13004
Yes
変換表のライトビットを1にセット
13005
No
領域の再割り当て必要か？
13006
Yes
全ての位置情報に対して
領域管理表より新規領域を割り当て
13007
位置情報をファイルシステムに報告
13008
ライトコマンド
発行
13009
コンピュータ102の
キャッシュディスク90
データの書き込み
13010
書き込み完了の報告
13011
書き込み完了
13012
複製要求
14000
ファイル履歴に最終更新日時と
複製日時を書き込む
13014
ライトビットを0にリセット
13015
終了
13016
終了
13017
終了
13018
更新失敗
13090
更新失敗
13091
更新失敗
13098
更新失敗
13099

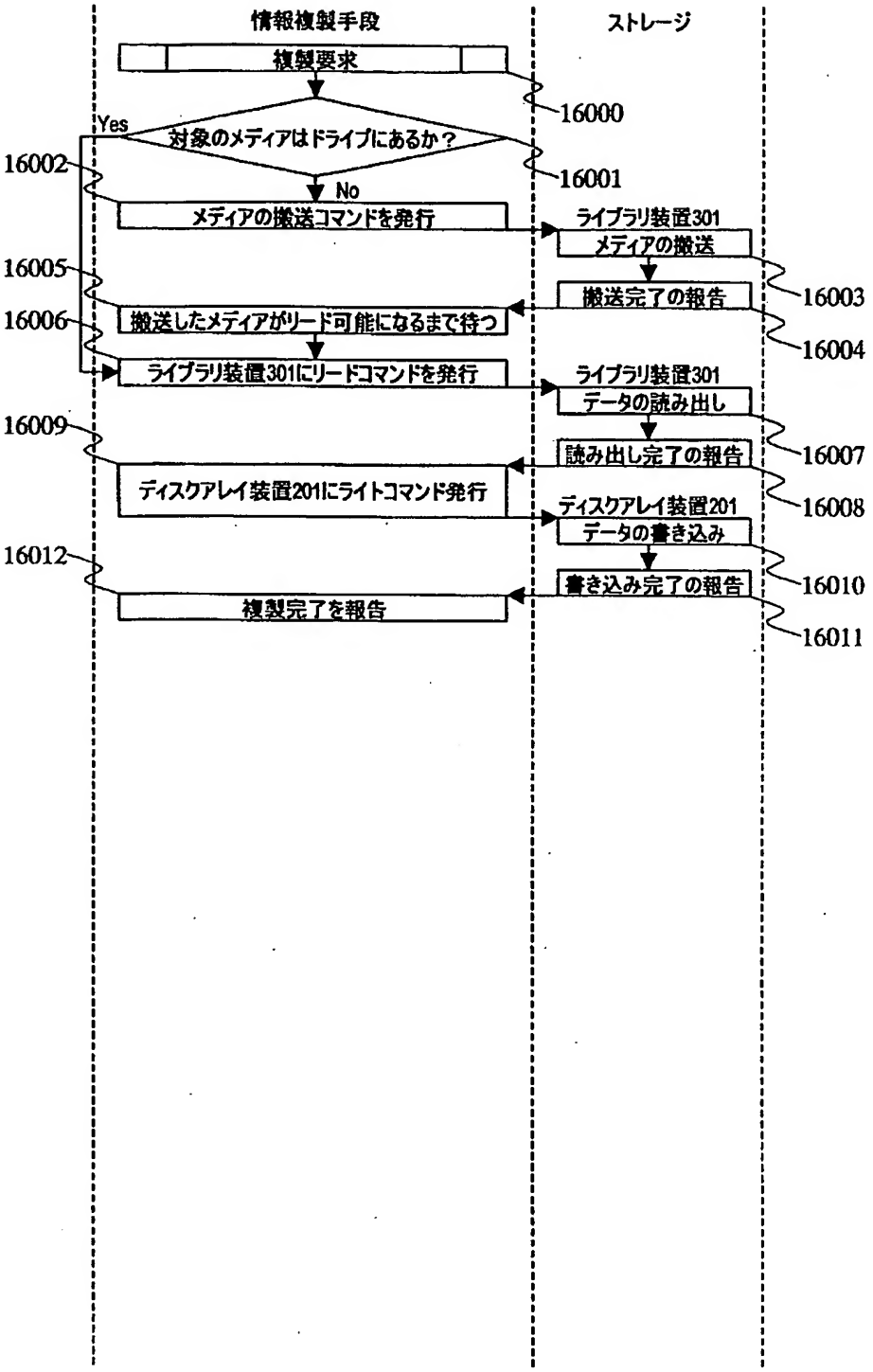
【図１４】


14000
情報複製手段
ストレージ
複製要求
コンピュータ102のキャッシュディスク90にリードコマンド発行
14001
コンピュータ102のキャッシュディスク90
データの読み出し
読み出し完了の報告
14002
14003
コンピュータ101のキャッシュディスク90とディスクアレイ装置201に対し同時にライトコマンドを発行
14004
コンピュータ101のキャッシュディスク90
ディスクアレイ装置201
データの書き込み
データの書き込み
14005
14005'
書き込み完了の報告
書き込み完了の報告
複製完了を報告
14006
14006'
14007

【図１５】


アプリケーション
ファイルシステム
位置管理手段
ストレージ
読み出し
15000
読み出し
15001
変換表より「AAA」に対応する行を選択
15002
No
ライトビットが0？
15003
Yes
読み出し失敗
15098
読み出し失敗
15099
リードカウントをインクリメント
15004
領域管理表より新規領域を割り当て
15005
対応する領域管理表のLBAを
割り当て済みに変更
15006
複製要求
16000
変換表に位置情報を書き込み
15007
有効ビットを1にセット
15008
位置情報をファイルシステムに報告
15009
リードコマンド
発行
15010
ディスクアレイ装置201
データの読み出し
15011
読み出し完了の報告
15012
読み出し完了
15013
ファイル履歴に最終アクセス日時と
複製日時を書き込み
15014
リードカウントをデクリメント
15015
終了
15016
終了
15017
終了
15018

【図１６】


情報複製手段
ストレージ
複製要求
16000
対象のメディアはドライブにあるか？
Yes
No
16001
16002
メディアの搬送コマンドを発行
ライブラリ装置301
メディアの搬送
16003
搬送完了の報告
16004
16005
搬送したメディアがリード可能になるまで待つ
16006
ライブラリ装置301にリードコマンドを発行
ライブラリ装置301
データの読み出し
16007
読み出し完了の報告
16008
16009
ディスクアレイ装置201にライトコマンド発行
ディスクアレイ装置201
データの書き込み
16010
書き込み完了の報告
16011
16012
複製完了を報告

フロントページの続き

(72)発明者　江口　賢哲
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内
(72)発明者　茂木　和彦
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内
(72)発明者　荒井　弘治
神奈川県小田原市国府津2880番地　株式会社日立製作所ストレージシステム事業部内
Fターム(参考)　5B082 EA01 EA07 EA09

OceanStore: An Architecture for Global-Scale Persistent Storage*

John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski,
Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea,
Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao
University of California, Berkeley
http://oceanstore.cs.berkeley.edu

ABSTRACT

OceanStore is a utility infrastructure designed to span the globe and provide continuous access to persistent information. Since this infrastructure is comprised of untrusted servers, data is protected through redundancy and cryptographic techniques. To improve performance, data is allowed to be cached anywhere, anytime. Additionally, monitoring of usage patterns allows adaptation to regional outages and denial of service attacks; monitoring also enhances performance through pro-active movement of data. A prototype implementation is currently under development.

1 INTRODUCTION

In the past decade we have seen astounding growth in the performance of computing devices. Even more significant has been the rapid pace of miniaturization and related reduction in power consumption of these devices. Based on these trends, many envision a world of ubiquitous computing devices that add intelligence and adaptability to ordinary objects such as cars, clothing, books, and houses. Before such a revolution can occur, however, computing devices must become so reliable and resilient that they are completely transparent to the user [50].

In pursuing transparency, one question immediately comes to mind: *where does persistent information reside?* Persistent information is necessary for transparency, since it permits the behavior of devices to be independent of the devices themselves, allowing an embedded component to be rebooted or replaced without losing vital configuration information. Further, the loss or destruction of a device does not lead to lost data. Note that a uniform infrastructure for accessing and managing persistent information can also provide for transparent synchronization among devices. Maintaining the consistency of these devices in the infrastructure allows users to safely access the same information from many different devices simultaneously [38]. Today, such sharing often requires laborious, manual synchronization.

Ubiquitous computing places several requirements on a persistent infrastructure. First, some form of (possibly intermittent) *connectivity* must be provided to computing devices, no matter how small. Fortunately, increasing levels of connectivity are being provided to consumers through cable-modems, DSL, cell-phones and wireless data services. Second, information must be kept *secure* from theft and denial-of-service (DoS). Since we assume wide-scale connectivity, we need to take extra measures to make sure that information is protected from prying eyes and malicious hands. Third, information must be extremely *durable*. Therefore changes should be submitted to the infrastructure at the earliest possible moment; sorting out the proper order for consistent commitment may come later. Further, archiving of information should be automatic and reliable.

Finally, *information* must be divorced from *location*. Centralized servers are subject to crashes, DoS attacks, and unavailability due to regional network outages. Although bandwidth in the core of the Internet has been doubling at a incredible rate, *latency* has not been improving as quickly. Further, connectivity at the leaves of the network is intermittent, of high latency, and of low bandwidth. Thus, to achieve *uniform* and *highly-available* access to information, servers must be geographically distributed and should exploit caching close to (or within) clients. As a result, we envision a model in which information is free to migrate to wherever it is needed, somewhat in the style of COMA shared memory multiprocessors [21].

As a rough estimate, we imagine providing service to roughly 10^{10} users, each with at least 10,000 files. OceanStore must therefore support over 10^{14} files.

1.1 OceanStore: a True Data Utility

We envision a cooperative utility model in which consumers pay a monthly fee in exchange for access to persistent storage. Such a utility should be highly-available from anywhere in the network, employ automatic replication for disaster recovery, use strong security by default, and provide performance that is similar to that of existing LAN-based networked storage systems under many circumstances. Services would be provided by a confederation of companies. Each user would pay their fee to one particular "utility provider", although they could consume storage and bandwidth resources from many different providers; providers would buy and sell capacity among themselves to make up the difference. Airports or small cafés could install servers on their premises to give customers better performance; in return they would get a small dividend for their participation in the global utility.

*This research is supported by NSF career award #ANI-9985250, DARPA grant #N66001-99-2-8913, and DARPA grant #DABT63-96-C-0056.

Patrick Eaton is supported by a National Defense Science and Engineering Graduate Fellowship (NDSEG); Dennis Geels is supported by the Fannie and John Hertz Foundation; and Hakim Weatherspoon is supported by an Intel Masters Fellowship.

ASPLOS 2000 Cambridge, MA Nov. 12-15, 2000

Ideally, a user would entrust all of his or her data to OceanStore; in return, the utility's economies of scale would yield much better availability, performance, and reliability than would be available otherwise. Further, the geographic distribution of servers would support *deep archival storage*, i.e. storage that would survive major disasters and regional outages. In a time when desktop workstations routinely ship with tens of gigabytes of spinning storage, the management of data is far more expensive than the storage media. OceanStore hopes to take advantage of this excess of storage space to make the management of data seamless and carefree.

1.2 Two Unique Goals

The OceanStore system has two design goals that differentiate it from similar systems: (1) the ability to be constructed from an *untrusted infrastructure* and (2) support of *nomadic data*.

Untrusted Infrastructure: OceanStore assumes that the infrastructure is fundamentally *untrusted*. Servers may crash without warning or leak information to third parties. This lack of trust is inherent in the utility model and is different from other cryptographic systems such as [35]. Only clients can be trusted with cleartext—all information that enters the infrastructure must be encrypted. However, rather than assuming that servers are passive repositories of information (such as in CFS [5]), we allow servers to be able to participate in protocols for distributed consistency management. To this end, we must assume that most of the servers are working correctly most of the time, and that there is one class of servers that we can trust to carry out protocols on our behalf (but not trust with the content of our data). This *responsible party* is financially responsible for the integrity of our data.

Nomadic Data: In a system as large as OceanStore, locality is of extreme importance. Thus, we have as a goal that data can be cached *anywhere*, *anytime*, as illustrated in Figure 1. We call this policy *promiscuous caching*. Data that is allowed to flow freely is called *nomadic data*. Note that nomadic data is an extreme consequence of separating information from its physical location. Although promiscuous caching complicates data coherence and location, it provides great flexibility to optimize locality and to trade off consistency for availability. To exploit this flexibility, continuous *introspective* monitoring is used to discover tacit relationships between objects. The resulting "meta-information" is used for locality management. Promiscuous caching is an important distinction between OceanStore and systems such as NFS [43] and AFS [23] in which cached data is confined to particular servers in particular regions of the network. Experimental systems such as XFS [3] allow "cooperative caching" [12], but only in systems connected by a fast LAN.

The rest of this paper is as follows: Section 2 gives a system-level overview of the OceanStore system. Section 3 shows sample applications of the OceanStore. Section 4 gives more architectural detail, and Section 5 reports on the status of the current prototype. Section 6 examines related work. Concluding remarks are given in Section 7.

2 SYSTEM OVERVIEW

An OceanStore prototype is currently under development. This section provides a brief overview of the planned system. Details on the individual system components are left to Section 4.

The fundamental unit in OceanStore is the *persistent object*. Each object is named by a *globally unique identifier*, or GUID.

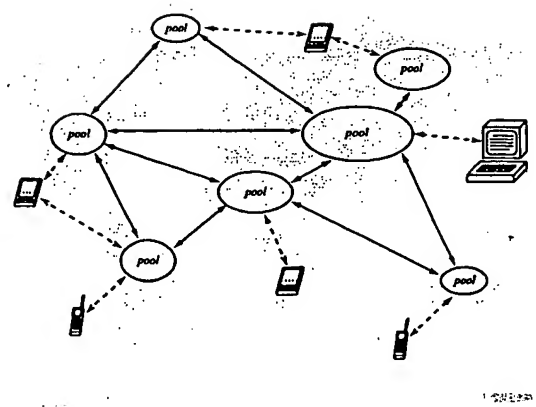


Figure 1: *The OceanStore system.* The core of the system is composed of a multitude of highly connected "pools", among which data is allowed to "flow" freely. Clients connect to one or more pools, perhaps intermittently.

Objects are replicated and stored on multiple servers. This replication provides availability[1] in the presence of network partitions and durability against failure and attack. A given replica is independent of the server on which it resides at any one time; thus we refer to them as *floating replicas*.

A replica for an object is located through one of two mechanisms. First, a fast, probabilistic algorithm attempts to find the object near the requesting machine. If the probabilistic algorithm fails, location is left to a slower, deterministic algorithm.

Objects in the OceanStore are modified through *updates*. Updates contain information about what changes to make to an object and the assumed state of the object under which those changes were developed, much as in the Bayou system [13]. In principle, every update to an OceanStore object creates a new version[2]. Consistency based on versioning, while more expensive to implement than update-in-place consistency, provides for cleaner recovery in the face of system failures [49]. It also obviates the need for backup and supports "permanent" pointers to information.

OceanStore objects exist in both *active* and *archival* forms. An active form of an object is the latest version of its data together with a handle for update. An archival form represents a permanent, read-only version of the object. Archival versions of objects are encoded with an erasure code and spread over hundreds or thousands of servers [18]; since data can be reconstructed from *any* sufficiently large subset of fragments, the result is that nothing short of a global disaster could ever destroy information. We call this highly redundant data encoding *deep archival storage*.

An application writer views the OceanStore as a number of sessions. Each session is a sequence of read and write requests related to one another through the session guarantees, in the style of the Bayou system [13]. Session guarantees dictate the level of consistency seen by a session's reads and writes; they can range from supporting extremely loose consistency semantics to supporting the ACID semantics favored in databases. In support of legacy code, OceanStore also provides an array of familiar interfaces such as the Unix file system interface and a simple transactional interface.

[1] If application semantics allow it, this availability is provided at the expense of consistency.

[2] In fact, groups of updates are combined to create new versions, and we plan to provide interfaces for retiring old versions, as in the Elephant File System [44].

Finally, given the flexibility afforded by the naming mechanism and to promote hands-off system maintenance, OceanStore exploits a number of dynamic optimizations to control the placement, number, and migration of objects. We classify all of these optimizations under the heading of *introspection*, an architectural paradigm that formalizes the automatic and dynamic optimization employed by "intelligent" systems.

3 APPLICATIONS

In this section we present applications that we are considering for OceanStore. While each of these applications can be constructed in isolation, OceanStore enables them to be developed more easily and completely by providing a single infrastructure for their shared, difficult problems. These problems include consistency, security, privacy, wide-scale data dissemination, dynamic optimization, durable storage, and disconnected operation. OceanStore solves these problems once, allowing application developers to focus on higher-level concerns.

One obvious class of applications for OceanStore is that of groupware and personal information management tools, such as calendars, email, contact lists, and distributed design tools. These applications are challenging to implement because they must allow for concurrent updates from many people. Further, they require that users see an ever-progressing view of shared information, even when conflicts occur. OceanStore's flexible update mechanism solves many of these problems. It provides ways to merge information and detect conflicts, as well as the infrastructure to disseminate information to all interested parties. Additionally, OceanStore provides ubiquitous access to data so that any device can access the information from anywhere.

Email is a particularly interesting groupware target for OceanStore. Although email applications appear mundane on the surface, their implementations are difficult because the obvious solution of filtering all messages through a single email server does not scale well, and distributed solutions have complicated internal consistency issues. For example, an email inbox may be simultaneously written by numerous different users while being read by a single user. Further, some operations, such as message move operations, must occur atomically even in the face of concurrent access from several clients to avoid data loss. In addition, email requires privacy and security by its very nature. OceanStore alleviates the need for clients to implement their own locking and security mechanisms, while enabling powerful features such as nomadic email collections and disconnected operation. Introspection permits a user's email to migrate closer to his client, reducing the round trip time to fetch messages from a remote server. OceanStore enables disconnected operation through its optimistic concurrency model—users can operate on locally cached email even when disconnected from the network; modifications are automatically disseminated upon reconnection.

In addition to groupware applications, OceanStore can be used to create very large digital libraries and repositories for scientific data. Both of these applications require massive quantities of storage, which in turn require complicated management. OceanStore provides a common mechanism for storing and managing these large data collections. It replicates data for durability and availability. Its deep archival storage mechanisms permit information to survive in the face of global disaster. Further, OceanStore benefits these applications by providing for seamless migration of data to where it is needed. For example, OceanStore can quickly disseminate vast streams of data from physics laboratories to the researchers around the world who analyze such data.

Finally, OceanStore provides an ideal platform for new streaming applications, such as sensor data aggregation and dissemination. Many have speculated about the utility of data that will emanate from the plethora of small MEMS sensors in the future; OceanStore provides a uniform infrastructure for transporting, filtering, and aggregating the huge volumes of data that will result.

4 SYSTEM ARCHITECTURE

In this section, we will describe underlying technologies that support the OceanStore system. We start with basic issues, such as *naming* and *access control*. We proceed with a description of the *data location* mechanism, which must locate objects anywhere in the world. Next, we discuss the OceanStore *update model* and the issues involved with *consistency management* in an untrusted infrastructure. After a brief word on the architecture for *archival storage*, we discuss the OceanStore API as presented to clients. Finally, we provide a description of the role of *introspection* in OceanStore.

4.1 Naming

At the lowest level, OceanStore objects are identified by a *globally unique identifier* (GUID), which can be thought of as a pseudo-random, fixed-length bit string. Users of the system, however, will clearly want a more accessible naming facility. To provide a facility that is both decentralized and resistant to attempts by adversaries to "hijack" names that belong to other users, we have adapted the idea of self-certifying path names due to Mazières [35].

An object GUID is the secure hash[3] of the owner's key and some human-readable name. This scheme allows servers to verify an object's owner efficiently, which facilitates access checks and resource accounting[4].

Certain OceanStore objects act as directories, mapping human-readable names to GUIDs. To allow arbitrary directory hierarchies to be built, we allow directories to contain pointers to other directories. A user of the OceanStore can choose several directories as "roots" and secure those directories through external methods, such as a public key authority. Note, however, that such root directories are only roots with respect to the clients that use them; the system as a whole has no one root. This scheme does not solve the problem of generating a secure GUID mapping, but rather reduces it to a problem of secure key lookup. We address this problem using the locally linked name spaces from the SDSI framework [1, 42].

Note that GUIDs identify a number of other OceanStore entities such as servers and archival fragments. The GUID for a server is a secure hash of its public key; the GUID for an archival fragment is a secure hash over the data it holds. As described in Section 4.3, entities in the OceanStore may be addressed directly by their GUID.

4.2 Access control

OceanStore supports two primitive types of access control, namely *reader* restriction and *writer* restriction. More complicated access control policies, such as working groups, are constructed from these two.

Restricting readers: To prevent unauthorized reads, we encrypt all data in the system that is not completely public and distribute the encryption key to those users with read permission. To revoke read permission, the owner must request that replicas be deleted or re-encrypted with the new key. A recently-revoked reader is able

[3] Our prototype system uses SHA-1 [37] for its secure hash.

[4] Note that each user might have more than one public key. They might also choose different public keys for private objects, public objects, and objects shared with various groups.

to read old data from cached copies or from misbehaving servers that fail to delete or re-key; however, this problem is not unique to OceanStore. Even in a conventional system, there is no way to force a reader to forget what has been read.

Restricting writers: To prevent unauthorized writes, we require that all writes be signed so that well-behaved servers and clients can verify them against an access control list (ACL). The owner of an object can securely choose the ACL x for an object *foo* by providing a signed certificate that translates to "Owner says use ACL x for object *foo*". The specified ACL may be another object or a value indicating a common default. An ACL entry extending privileges must describe the privilege granted and the signing key, but not the explicit identity, of the privileged users. We make such entries publicly readable so that servers can check whether a write is allowed. We plan to adopt ideas from systems such as Taos and PolicyMaker to allow users to express and reason formally about a wide range of possible policies [52, 6].

Note the asymmetry that has been introduced by encrypted data: reads are restricted at clients via key distribution, while writes are restricted at servers by ignoring unauthorized updates.

4.3 Data Location and Routing

Entities in the OceanStore are free to reside on *any* of the OceanStore servers. This freedom provides maximum flexibility in selecting policies for replication, availability, caching, and migration. Unfortunately, it also complicates the process of locating and interacting with these entities. Rather than restricting the placement of data to aid in the location process, OceanStore tackles the problem of data location head-on. The paradigm is that of query routing, in which the network takes an active role in routing messages to objects.

4.3.1 *Distributed Routing in OceanStore*

Every addressable entity in the OceanStore (*e.g.* floating replica, archival fragment, or client) is identified by one or more GUIDs. Entities that are functionally equivalent, such as different replicas for the same object, are identified by the same GUID. Clients interact with these entities with a series of protocol messages, as described in subsequent sections. To support location-independent addressing, OceanStore messages are labeled with a destination GUID, a random number, and a small predicate. The destination IP address does not appear in these messages. The role of the OceanStore routing layer is to route messages directly to the closest node that matches the predicate and has the desired GUID.

In order perform this routing process, the OceanStore networking layer consults a distributed, fault-tolerant data structure that explicitly tracks the location of all objects. Routing is thus a two phase process. Messages begin by routing from node to node along the distributed data structure until a destination is discovered. At that point, they route directly to the destination. It is important to note that the OceanStore routing layer does not supplant IP routing, but rather provides additional functionality on top of IP.

There are many advantages to combining data location and routing in this way. First and foremost, the task of routing a particular message is handled by the aggregate resources of many different nodes. By exploiting multiple routing paths to the destination, this serves to limit the power of compromised nodes to deny service to a client. Second, messages route directly to their destination, avoiding the multiple round-trips that a separate data location and routing process would incur. Finally, the underlying infrastructure has more up-to-date information about the current location of entities than the clients. Consequently, the combination of location and routing permits communication with "the closest" entity, rather than an entity that the client might have heard of in the past. If replicas move around, only the network, not the users of the data, needs to know.

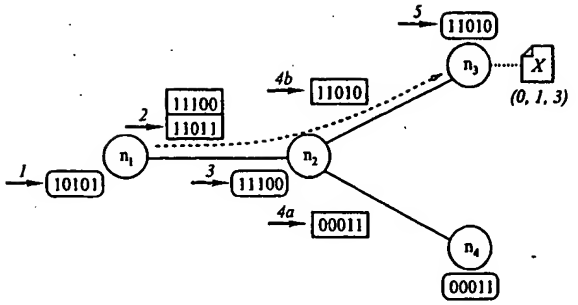


Figure 2: *The probabilistic query process.* The replica at n_1 is looking for object X, whose GUID hashes to bits 0, 1, and 3. (1) The local Bloom filter for n_1 (rounded box) shows that it does not have the object, but (2) its neighbor filter (unrounded box) for n_2 indicates that n_2 might be an intermediate node en route to the object. The query moves to n_2, (3) whose Bloom filter indicates that it does not have the document locally, (4a) that its neighbor n_4 doesn't have it either, but (4b) that its neighbor n_3 might. The query is forwarded to n_3, (5) which verifies that it has the object.

The mechanism for routing is a two-tiered approach featuring a fast, probabilistic algorithm backed up by a slower, reliable hierarchical method. The justification for this two-level hierarchy is that entities that are accessed frequently are likely to reside close to where they are being used; mechanisms to ensure this locality are described in Section 4.7. Thus, the probabilistic algorithm routes to entities rapidly if they are in the local vicinity. If this attempt fails, a large-scale hierarchical data structure in the style of Plaxton et. al. [40] locates entities that cannot be found locally. We will describe these two techniques in the following sections.

4.3.2 *Attenuated Bloom Filters*

The probabilistic algorithm is fully distributed and uses a constant amount of storage per server. It is based on the idea of hill-climbing; if a query cannot be satisfied by a server, local information is used to route the query to a likely neighbor. A modified version of a Bloom filter [7]—called an *attenuated* Bloom filter—is used to implement this potential function.

An attenuated Bloom filter of depth D can be viewed as an array of D normal Bloom filters. In the context of our algorithm, the first Bloom filter is a record of the objects contained locally on the current node. The ith Bloom filter is the union of all of the Bloom filters for all of the nodes a distance i through any path from the current node. An attenuated Bloom filter is stored for each directed edge in the network. A query is routed along the edge whose filter indicates the presence of the object at the smallest distance. This process is illustrated in Figure 2. Our current metric of distance is hop-count, but in the future we hope to include a more precise measure corresponding roughly to latency. Also, "reliability factors" can be applied locally to increase the distance to nodes that have abused the protocol in the past, automatically routing around certain classes of attacks.

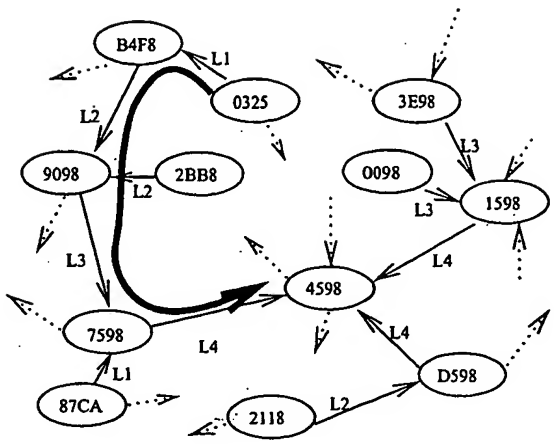


Figure 3: A portion of the global mesh, rooted at node 4598. Paths from any node to the root of any tree can be traversed by resolving the root's ID one digit at a time; the bold arrow shows a route from node 0325 to node 4598. Data location uses this structure. Note that most object searches do not travel all the way to the root (see text).

4.3.3 *The Global Algorithm: Wide-scale Distributed Data Location*

The global algorithm for the OceanStore is a variation on Plaxton et. al.'s randomized hierarchical distributed data structure [40], which embeds multiple random trees in the network. Although OceanStore uses a highly-redundant version of this data structure, it is instructive to understand the basic Plaxton scheme. In that scheme, every server in the system is assigned a random (and unique) node-ID. These node-IDs are then used to construct a mesh of neighbor links, as shown in Figure 3. In this figure, each link is labeled with a level number that denotes the stage of routing that uses this link. In the example, the links are constructed by taking each node-ID and dividing it into chunks of four bits. The N^{th} level neighbor-links for some Node X point at the 16 *closest neighbors*[5] whose node-IDs match the lowest N-1 nibbles of Node X's ID and who have different combinations of the N^{th} nibble; one of these links is always a loopback link. If a link cannot be constructed because no such node meets the proper constraints, then the scheme chooses the node that matches the constraints as closely as possible. This process is repeated for all nodes and levels within a node.

The key observation to make from Figure 3 is that the links form a series of random embedded trees, with each node as the root of one of these trees. As a result, the neighbor links can be used to route from anywhere to a given node, simply by resolving the node's address one link at a time—first a level-one link, then a level-two link, etc. To use this structure for data location, we map each object to a single node whose node-ID matches the object's GUID in the most bits (starting from the least significant); call this node the object's *root*. If information about the GUID (such as its location) were stored at its root, then anyone could find this information simply by following neighbor links until they reached the root node for the GUID. As described, this scheme has nice load distribution properties, since GUIDs become randomly mapped throughout the infrastructure.

This random distribution would appear to reduce locality; however, the Plaxton scheme achieves locality as follows: when a replica is placed somewhere in the system, its location is "published" to the routing infrastructure. The publishing process works its way to the object's root and deposits a pointer at every hop along the way. This process requires $O(\log n)$ hops, where n is the number of servers in the world. When someone searches for information, they climb the tree until they run into a pointer, after which they route directly to the object. In [40], the authors show that the average distance traveled is proportional to the distance between the source of the query and the closest replica that satisfies this query.

Achieving Fault Tolerance: The basic scheme described above is sensitive to a number of different failures. First, each object has a single root, which becomes a single point of failure, the potential subject of denial of service attacks, and an availability problem. OceanStore addresses this weakness in a simple way: it hashes each GUID with a small number of different salt values. The result maps to several different root nodes, thus gaining redundancy and simultaneously making it difficult to target a single node with a denial of service attack against a range of GUIDs.

A second problem with the above scheme is sensitivity to corruption in the links and pointers. An important observation, however, is that the above structure has sufficient redundancy to tolerate small amounts of corruption. Bad links can be immediately detected, and routing can be continued by jumping to a random neighbor node[6]. To increase this redundancy, the OceanStore location structure supplements the basic links of the above scheme with additional neighbor links. Further, the infrastructure continually monitors and repairs neighbor links (a form of *introspection*—see Section 4.7), and servers slowly repeat the publishing process to repair pointers.

The Advantages of Distributed Information: The advantages of a Plaxton-like data structure in the OceanStore are many. First, it is a highly redundant and fault-tolerant structure that spreads data location load evenly while finding local objects quickly. The combination of the probabilistic and global algorithms should comfortably scale to millions of servers. Second, the aggregate information contained in this data structure is sufficient to recognize which servers are down and to identify data that must be reconstructed when a server is permanently removed. This feature is important for maintaining a minimum level of redundancy for the deep archival storage. Finally, the Plaxton links form a natural substrate on which to perform network functions such as admission control and multicast.

Achieving Maintenance-Free Operation: While existing work on Plaxton-like data structures did not include algorithms for online creation and maintenance of the global mesh, we have produced recursive node insertion and removal algorithms. These make use of the redundant neighbor links mentioned above. Further, we have generalized our publication algorithm to support replicated roots, which remove single-points of failure in data location. Finally, we have optimized failure modes by using soft-state beacons to detect faults more quickly, time-to-live fields to react better to routing updates, and a second-chance algorithm to minimize the cost of recovering lost nodes. This information is coupled with continuous repair mechanisms that recognize when

[5] "Closest" means with respect to the underlying IP routing infrastructure. Roughly speaking, the measurement metric is the time to route via IP.

[6] Each tree spans every node, hence any node should be able to reach the root.

servers have been down for a long time and need to have their data reconstructed[7]. The practical implication of this work is that the OceanStore infrastructure as a whole automatically adapts to the presence or absence of particular servers without human intervention, greatly reducing the cost of management.

4.4 Update Model

Several of the applications described in Section 3 exhibit a high degree of write sharing. To allow for concurrent updates while avoiding many of the problems inherent with wide-area locking, OceanStore employs an update model based on *conflict resolution*. Conflict resolution was introduced in the Bayou system [13] and supports a range of consistency semantics—up to and including ACID semantics. Additionally, conflict resolution reduces the number of aborts normally seen in detection-based schemes such as optimistic concurrency control [29].

Although flexible, conflict resolution requires the ability to perform server-side computations on data. In an untrusted infrastructure, replicas have access only to ciphertext, and no one server is trusted to perform commits. Both of these issues complicate the update architecture. However, the current OceanStore design is able to handle many types of conflict resolution directly on encrypted data. The following paragraphs describe the issues involved and our progress towards solving them.

4.4.1 Update Format and Semantics

Changes to data objects within OceanStore are made by client-generated *updates*, which are lists of predicates associated with actions. The semantics of an update are as follows: to apply an update against a data object, a replica evaluates each of the update's predicates in order. If any of the predicates evaluates to true, the actions associated with the earliest true predicate are atomically applied to the data object, and the update is said to *commit*. Otherwise, no changes are applied, and the update is said to *abort*. The update itself is logged regardless of whether it commits or aborts.

Note that OceanStore update semantics are similar to those of the Bayou system, except that we have eliminated the merge procedure used there, since arbitrary computations and manipulations on ciphertext are still intractable. Nevertheless, we preserve the key functionality of their model, which they found to be expressive enough for a number of sample applications including a group calendar, a shared bibliographic database, and a mail application [14]. Furthermore, the model can be applied to other useful applications. For instance, Coda [26] provided specific merge procedures for conflicting updates of directories; this type of conflict resolution is easily supported under our model. Slight extensions to the model can support Lotus Notes-style conflict resolution, where unresolvable conflicts result in a branch in the object's version stream [25]. Finally, the model can be used to provide ACID semantics: the first predicate is made to check the read set of a transaction, the corresponding action applies the write set, and there are no other predicate-action pairs.

4.4.2 Extending the Model to Work over Ciphertext

OceanStore replicas are not trusted with unencrypted information. This complicates updates by restricting the set of predicates that replicas can compute and the set of actions they are able to apply. However, the following predicates are currently possible: *compare-version*, *compare-size*, *compare-block*, and *search*. The first two predicates are trivial since they are over the unencrypted meta-data

[7] Note that the read-only nature of most of the information in the OceanStore makes this reconstruction particularly easy; see Section 4.5.

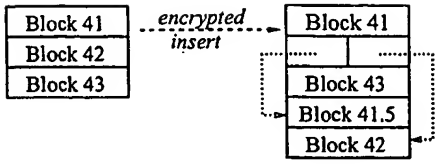


Figure 4: *Block insertion on ciphertext.* The client wishes to insert block 41.5, so she appends it and block 42 to the object, then replaces the old block 42 with a block pointing to the two appended blocks. The server learns nothing about the contents of any of the blocks.

of the object. The *compare-block* operation is easy if the encryption technology is a position-dependent block cipher: the client simply computes a hash of the encrypted block and submits it along with the block number for comparison. Perhaps the most impressive of these predicates is *search*, which can be performed directly on ciphertext [47]; this operation reveals only that a search was performed along with the boolean result. The cleartext of the search string is not revealed, nor can the server initiate new searches on its own.

In addition to these predicates, the following operations can be applied to ciphertext: *replace-block*, *insert-block*, *delete-block*, and *append*. Again assuming a position-dependent block cipher, the *replace-block* and *append* operations are simple for the same reasons as *compare-block*.

The last two operations, *insert-block* and *delete-block*, can be performed by grouping blocks of the object into two sets, index blocks and data blocks, where index blocks contain pointers to other blocks elsewhere in the object. To insert, one replaces the block at the insertion point with a new block that points to the old block and the inserted block, both of which are appended to the object. This scheme is illustrated in Figure 4. To delete, one replaces the block in question with an empty pointer block. Note that this scheme leaks a small amount of information and thus might be susceptible to compromise by a traffic-analysis attack; users uncomfortable with this leakage can simply append encrypted log records to an object and rely on powerful clients to occasionally generate and re-encrypt the object in whole from the logs.

The schemes presented in this section clearly impact the format of objects. However, these schemes are the subject of ongoing research; more flexible techniques will doubtless follow.

4.4.3 Serializing Updates in an Untrusted Infrastructure

The process of conflict resolution starts with a series of updates, chooses a total order among them, then applies them atomically in that order. The easiest way to compute this order is to require that all updates pass through a *master* replica. Unfortunately, trusting any one replica to perform this task is incompatible with the untrusted infrastructure assumption on which OceanStore is built. Thus, we replace this master replica with a *primary tier* of replicas. These replicas cooperate with one another in a Byzantine agreement protocol [30] to choose the final commit order for updates[8]. A secondary tier of replicas communicates among themselves and the primary tier via an enhanced *epidemic* algorithm, as in Bayou.

The decision to use two classes of floating replicas is motivated by several considerations. First, all known protocols that are toler-

[8] A Byzantine agreement protocol is one in which we assume that no more than m of the total $n = 3m + 1$ replicas are faulty.

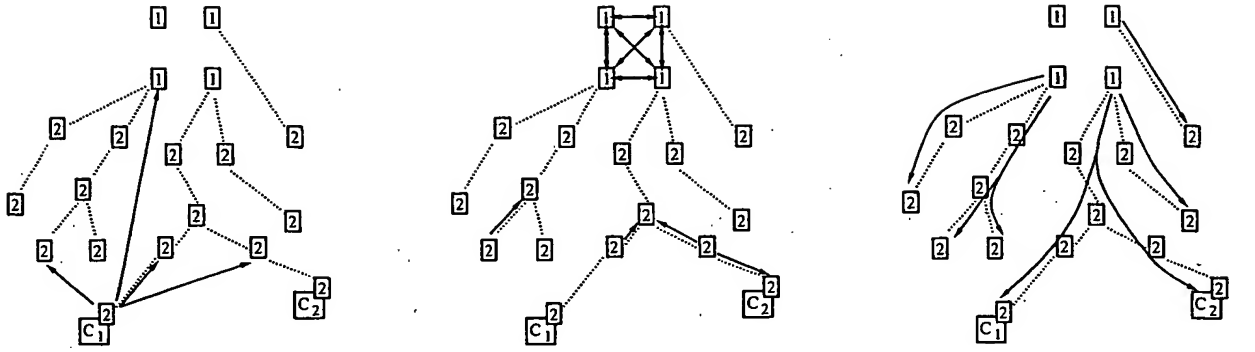


Figure 5: *The path of an update.* (a) After generating an update, a client sends it directly to the object's primary tier, as well as to several other random replicas for that object. (b) While the primary tier performs a Byzantine agreement protocol to commit the update, the secondary replicas propagate the update among themselves epidemically. (c) Once the primary tier has finished its agreement protocol, the result of the update is multicast down the dissemination tree to all of the secondary replicas.

ant to arbitrary replica failures are too communication-intensive to be used by more than a handful of replicas. The primary tier thus consists of a small number of replicas located in high-bandwidth, high-connectivity regions of the network[9]. To allow for later, off-line verification by a party who did not participate in the protocol, we are exploring the use of *proactive* signature techniques [4] to certify the result of the serialization process. We hope to extend the protocol in [10] to use such techniques.

Some applications may gain performance or availability by requiring a lesser degree of consistency than ACID semantics. These applications motivate the secondary tier of replicas in OceanStore. Secondary replicas do not participate in the serialization protocol, may contain incomplete copies of an object's data, and can be more numerous than primary replicas. They are organized into one or more application-level multicast trees, called *dissemination trees*, that serve as conduits of information between the primary tier and secondary tier. Among other things, the dissemination trees *push* a stream of committed updates to the secondary replicas, and they serve as communication paths along which secondary replicas *pull* missing information from parents and primary replicas. This architecture permits dissemination trees to transform *updates* into *invalidations* as they progress downward; such a transformation is exploited at the leaves of the network where bandwidth is limited.

Secondary replicas contain both tentative[10] and committed data. They employ an epidemic-style communication pattern to quickly spread tentative commits among themselves and to pick a tentative serialization order. To increase the chances that this tentative order will match the final ordering chosen by the primary replicas, clients optimistically timestamp their updates. Secondary replicas order tentative updates in timestamp order, and the primary tier uses these same timestamps to guide its ordering decisions. Since the serialization decisions of the secondary tier are tentative, they may be safely decided by untrusted replicas; applications requiring stronger consistency guarantees must simply wait for their updates to reach the primary tier.

4.4.4 *A Direct Path to Clients and Archival Storage*

The full path of an update is shown in Figure 5. Note that this path is optimized for low latency and high throughput. Under ideal circumstances, updates flow directly from the client to the primary tier of servers, where they are serialized and then multicast to the secondary servers. All of the messages shown here are addressed through GUIDs, as described in Section 4.3. Consequently, the update protocol operates entirely without reference to the physical location of replicas.

One important aspect of OceanStore that differs from existing systems is the fact that the archival mechanisms are tightly coupled with update activity. After choosing a final order for updates, the inner tier of servers signs the result and sends it through the dissemination tree. At the same time, these servers generate encoded, archival fragments and distribute them widely. Consequently, updates are made extremely durable as a direct side-effect of the commitment process. Section 4.5 discusses archival storage in more detail.

4.4.5 *Efficiency of the Consistency Protocol*

There are two main points of interest when considering the efficiency of the consistency protocol: the amount of network bandwidth the protocol demands, and the latency between when an update is created and when the client receives notification that it has committed or aborted. Assuming that a Byzantine agreement protocol like that in [10] is used, the total cost an update in bytes sent across the network, b, is given by the equation:

$$b = c_1 n^2 + (u + c_2)n + c_3$$

where u is the size of the update, n is the number of replicas in the primary tier, and c_1, c_2, and c_3 are the sizes of small protocol messages. While this equation appears to be dominated by the n^2 term, the constant c_1 is quite small, on the order of 100 bytes. Thus for sufficiently small n and large updates, the equation is dominated by the n term. Since there are n replicas, the minimum amount of bytes that must be transfered to keep all replicas up to date is un.

Figure 6 shows the cost of an update, normalized to this minimum amount, as a function of update size. Note that for $m = 4$ and $n = 13$, the normalized cost approaches 1 for update sizes around 100k bytes, but it approaches 2 at update sizes of only around 4k bytes.[11] Thus for updates of 4k bytes or more, our system uses less than double the minimum amount of network bandwidth necessary to keep all the replicas in the primary tier up to date.

[9] The choice of which replicas to include in the primary tier is left to the client's responsible party, which must ensure that its chosen group satisfies the Byzantine assumption mentioned above.

[10] Tentative data is data that the primary replicas have not yet committed.

[11] Recall that m is the number of faulty replicas tolerated by the Byzantine agreement protocol.

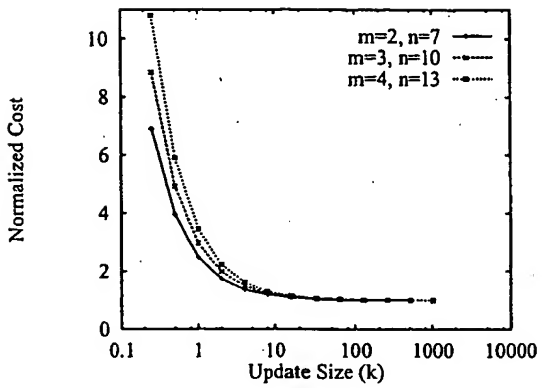


Figure 6: The cost of an update in bytes sent across the network, normalized to the minimum cost needed to send the update to each of the replicas.

Unfortunately, latency estimates for the consistency protocol are more difficult to come by without a functioning prototype. For this reason, let us suffice it to say that there are six phases of messages in the protocol we have described. Assuming latency of messages over the wide area dominates computation time and that each message takes 100ms, we have an approximate latency per update of less than a second. We believe this latency is reasonable, but we will need to complete our prototype system before we can verify the accuracy of this rough estimate.

4.5 Deep Archival Storage

The archival mechanism of OceanStore employs *erasure codes*, such as interleaved Read-Solomon codes [39] and Tornado codes [32]. Erasure coding is a process that treats input data as a series of fragments (say n) and transforms these fragments into a greater number of fragments (say $2n$ or $4n$). As mentioned in Section 4.4, the fragments are generated in parallel by the inner tier of servers during the commit process. The essential property of the resulting code is that *any* n of the coded fragments are sufficient to construct the original data[12].

Assuming that we spread coded fragments widely, it is very unlikely that enough servers will be down to prevent the recovery of data. We call this argument *deep archival storage*. A simple example will help illustrate this assertion. Assuming uncorrelated faults among machines, one can calculate the reliability at a given instant of time according to the following formula:

$$P = \sum_{i=0}^{r_f} \frac{\binom{m}{i}\binom{n-m}{f-i}}{\binom{n}{f}}$$

where P is the probability that a document is available, n is the number of machines, m is the number of currently unavailable machines, f is the number of fragments per document, and r_f is the maximum number of unavailable fragments that still allows the document to be retrieved. For instance, with a million machines, ten percent of which are currently down, simple replication without erasure codes provides only two nines (0.99) of reliability. A 1/2-rate erasure coding of a document into 16 fragments gives the document over five nines of reliability (0.999994), yet consumes the same amount of storage. With 32 fragments, the reliability increases by another factor of 4000, supporting the assertion that *fragmentation increases reliability*. This is a consequence of the law of large numbers.

To preserve the erasure nature of the fragments (meaning that a fragment is either retrieved correctly and completely, or not at all), we use a hierarchical hashing method to verify each fragment. We generate a hash over each fragment, and recursively hash over the concatenation of pairs of hashes to form a binary tree. Each fragment is stored along with the hashes neighboring its path to the root. When it is retrieved, the requesting machine may recalculate the hashes along that path. We can use the top-most hash as the GUID to the immutable archival object, making every fragment in the archive completely self-verifying.

For the user, we provide a naming syntax which explicitly incorporates version numbers. Such names can be included in other documents as a form of permanent hyper-link. In addition, interfaces will exist to examine modification history and to set versioning policies [44]. Although in principle every version of every object is archived, clients can choose to produce versions less frequently. Archival copies are also produced when objects are idle for a long time or before objects become inactive. When generating archival fragments, the floating replicas of an object participate together: they each generate a disjoint subset of the fragments and disseminate them into the infrastructure.

To maximize the survivability of archival copies, we identify and rank administrative domains by their reliability and trustworthiness. We avoid dispersing all of our fragments to locations that have a high correlated probability of failure. Further, the number of fragments (and hence the durability of information) is determined on a per-object basis. OceanStore contains processes that slowly sweep through all existing archival data, repairing or increasing the level of replication to further increase durability.

To reconstruct archival copies, OceanStore sends out a request keyed off the GUID of the archival versions. Note that we can make use of excess capacity to insulate ourselves from slow servers by requesting more fragments than we absolutely need and reconstructing the data as soon as we have enough fragments. As the request propagates up the location tree (Section 4.3), fragments are discovered and sent to the requester. This search has nice locality properties since closer fragments tend to be discovered first.

4.6 The OceanStore API

OceanStore draws much strength from its global scale, wide distribution, epidemic propagation method, and flexible update policy. The system as a whole can have rather complicated behavior. However, the OceanStore application programming interface (API) enables application writers to understand their interaction with the system.

This base API provides full access to OceanStore functionality in terms of sessions, session guarantees, updates, and callbacks. A session is a sequence of reads and writes to potentially different objects that are related to one another through session guarantees. Guarantees define the level of consistency seen by accesses through a session. The API provides mechanisms to develop arbitrarily complex updates in the form described in Section 4.4. The API also provides a callback feature to notify applications of relevant events. An application can register an application-level handler to be invoked at the occurrence of relevant events, such as the commit or abort of an update.

Applications with more basic requirements are supported through facades to the standard API. A facade is an interface to the API that provides a traditional, familiar interface. For example, a transaction facade would provide an abstraction atop the OceanStore API so that the developer could access the system

[12] Tornado codes, which are faster to encode and decode, require slightly more than n fragments to reconstruct the information.

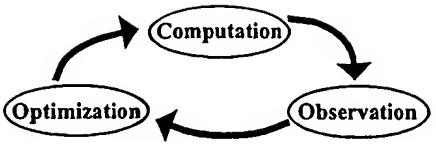


Figure 7: The Cycle of Introspection

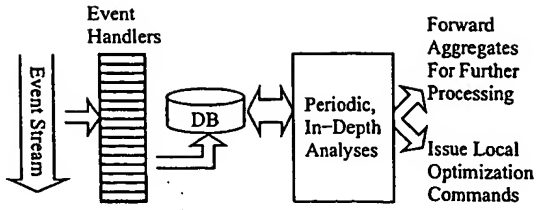


Figure 8: Fast event handlers summarize and respond to local events. For efficiency, the "database" may be only soft state (see text). Further processing analyzes trends and aggregate information across nodes.

in terms of traditional transactions. The facade would simplify the application writer's job by ensuring proper session guarantees, reusing standard update templates, and automatically computing read sets and write sets for each update.

Of course, OceanStore is a new system in a world of legacy code, and it would be unreasonable to expect the authors of existing applications to port their work to an as yet undeployed system. Therefore, OceanStore provides a number of legacy facades that implement common APIs, including a Unix file system, a transactional database, and a gateway to the World Wide Web. These interfaces exist as libraries or "plugins" to existing browsers or operating systems. They permit users to access legacy documents while enjoying the ubiquitous and secure access, durability, and performance advantages of OceanStore.

4.7 Introspection

As envisioned, OceanStore will consist of millions of servers with varying connectivity, disk capacity, and computational power. Servers and devices will connect, disconnect, and fail sporadically. Server and network load will vary from moment to moment. Manually tuning a system so large and varied is prohibitively complex. Worse, because OceanStore is designed to operate using the utility model, manual tuning would involve cooperation across administrative boundaries.

To address these problems, OceanStore employs *introspection*, an architectural paradigm that mimics adaptation in biological systems. As shown in Figure 7, introspection augments a system's normal operation (*computation*), with *observation* and *optimization*. *Observation modules* monitor the activity of a running system and keep a historical record of system behavior. They also employ sophisticated analyses to extract patterns from these observations. *Optimization modules* use the resulting analysis to adjust or adapt the computation.

OceanStore uses introspective mechanisms throughout the system. Although we have insufficient space to describe each use in detail, we will give a flavor of our techniques below.

4.7.1 Architecture

We have designed a common architecture for introspective systems in OceanStore (see Figure 8). These systems process local events, forwarding summaries up a distributed hierarchy to form approximate global views of the system. Events include any incoming message or noteworthy physical measurement. Our three-point approach provides a framework atop which we are developing specific observation and optimization modules.

The high event rate[13] precludes extensive online processing. Instead, a level of fast event handlers summarizes local events. These summaries are stored in a local database. At the leaves of the hierarchy, this database may reside only in memory; we loosen durability restrictions for local observations in order to attain the necessary event rate.

[13] Each machine initiates and receives roughly as many messages as local area network files systems. In addition, the routing infrastructure requires communication proportional to the logarithm of the size of the network.

We describe all event handlers in a simple domain-specific language. This language includes primitives for operations like averaging and filtering, but explicitly prohibits loops. We expect this model to provide sufficient power, flexibility, and extensibility, while enabling the verification of security and resource consumption restrictions placed on event handlers.

A second level of more powerful algorithms periodically processes the information in the database. This level can perform sophisticated analyses and incorporate historical information, allowing the system to detect and respond to long-term trends.

Finally, after processing and responding to its own events, a third level of each node forwards an appropriate summary of its knowledge to a parent node for further processing on the wider scale. The infrastructure uses the standard OceanStore location mechanism to locate that node, which is identified by its GUID. Conversely, we could distribute the information to remote optimization modules as OceanStore objects that would also be accessed via the standard location mechanism.

4.7.2 Uses of Introspection

We use introspection to manage a number of subsystems in the OceanStore. Below, we will discuss several of these components.

Cluster Recognition: Cluster recognition attempts to identify and group closely related files. Each client machine contains an event handler triggered by each data object access. This handler incrementally constructs a graph representing the semantic distance [28] among data objects, which requires only a few operations per access.

Periodically, we run a clustering algorithm that consumes this graph and detects clusters of strongly-related objects. The frequency of this operation adapts to the stability of the input and the available processing resources. The result of the clustering algorithm is forwarded to a global analysis layer that publishes small objects describing established clusters. Like directory listings, these objects help remote optimization modules collocate and prefetch related files.

Replica Management: Replica management adjusts the number and location of floating replicas in order to service access requests more efficiently. Event handlers monitor client requests and system load, noting when access to a specific replica exceeds its resource allotment. When access requests overwhelm a replica, it forwards a request for assistance to its parent node. The parent, which tracks locally available resources, can create additional floating replicas on nearby nodes to alleviate load.

Conversely, replica management eliminates floating replicas that have fallen into disuse. Notification of a replica's termination also

propagates to parent nodes, which can adjust that object's dissemination tree.

In addition to these short-term decisions, nodes regularly analyze global usage trends, allowing additional optimizations. For example, OceanStore can detect periodic migration of clusters from site to site and prefetch data based on these cycles. Thus users will find their project files and email folder on a local machine during the work day, and waiting for them on their home machines at night.

Other Uses: OceanStore uses introspective mechanisms in many other aspects as well. Specifically, introspection improves the manageability and performance of the routing structure, enables construction of efficient update dissemination trees, ensures the availability and durability of archival fragments, identifies unreliable peer organizations, and performs continuous confidence estimation on its own optimizations in order to reduce harmful changes and feedback cycles.

5 STATUS

We are currently implementing an OceanStore prototype that we will deploy for testing and evaluation. The system is written in Java with a state machine-based request model for fast I/O [22]. Initially, OceanStore will communicate with applications through a UNIX file system interface and a read-only proxy for the World Wide Web in addition to the native OceanStore API.

We have explored the requirements that our security guarantees place on a storage architecture. Specifically, we have explored differences between enforcing read and write permissions in an untrusted setting, emphasizing the importance of the ability of clients to validate the correctness of any data returned to them. This exploration included not only checking the integrity of the data itself, but also checking that the data requested was the data returned, and that all levels of metadata were protected as strongly as the data itself. A prototype cryptographic file system provided a testbed for specific security mechanisms.

A prototype for the probabilistic data location component has been implemented and verified. Simulation results show that our algorithm finds nearby objects with near-optimal efficiency.

We have implemented prototype archival systems that use both Reed-Solomon and Tornado codes for redundancy encoding. Although only one half of the fragments were required to reconstruct the object, we found that issuing requests for extra fragments proved beneficial due to dropped requests.

We have implemented the introspective prefetching mechanism for a local file system. Testing showed that the method correctly captured high-order correlations, even in the presence of noise. We will combine that mechanism with an optimization module appropriate for the wide-area network.

6 RELATED WORK

Distributed systems such as Taos [52] assume untrusted networks and applications, but rely on some trusted computing base. Cryptographic file systems such as Blaze's CFS [5] provide end-to-end secrecy, but include no provisions for sharing data, nor for protecting integrity independently from secrecy. The Secure File System [24] supports sharing with access control lists, but fails to provide independent support for integrity, and trusts a single server to distribute encryption keys. The Farsite project [8] is more similar to OceanStore than these other works, but while it assumes the use of untrusted clients, it does not address a wide-area infrastructure.

SDSI [1] and SPKI [15] address the problem of securely distributing keys and certificates in a decentralized manner. PolicyMaker [6] deals with the description of trust relations. Mazières proposes self-certifying paths to separate key management from system security [35].

Bloom filters [7] are commonly used as compact representations of large sets. The R* distributed database [33] calculates them on demand to implement efficient semijoins. The Summary Cache [16] pushes Bloom filters between cooperating web caches, although their method does not scale well in the number of caches.

Distributing data for performance, availability, or survivability has been studied extensively in both the file systems and database communities. A summary of distributed file systems can be found in [31]. In particular, Bayou [13] and Coda [26] use replication to improve availability at the expense of consistency and introduce specialized conflict resolution procedures. Sprite [36] also uses replication and caching to improve availability and performance, but has a guarantee of consistency that incurs a performance penalty in the face of multiple writers. None of these systems addresses the range of security concerns that OceanStore does, although Bayou examines some problems that occur when replicas are corrupted [48].

Gray et. al. argue against promiscuous replication in [19]. OceanStore differs from the class of systems they describe because it does not bind floating replicas to specific machines, and it does not replicate all objects at each server.

OceanStore's second tier of floating replicas are similar to transactional caches; in the taxonomy of [17] our algorithm is detection-based and performs its validity checks at commit time. In contrast to similar systems, our merge predicates should decrease the number of transactions aborted due to out-of-date caches.

Many previous projects have explored feedback-driven adaptation in extensible operating systems [45], databases [11], file systems [34], global operating systems [9], and storage devices [51]. Although these projects employ differing techniques and terminology, each could be analyzed with respect to the *introspective* model.

The Seer project formulated the concept of semantic distance [28] and collects clusters of related files for automated hoarding. Others have used file system observation to drive automatic prefetching [20, 27].

Introspective replica management for web content was examined in AT&T's Radar project [41], which considers read-only data in a trusted infrastructure. The Mariposa project [46] addresses inter-domain replication with an economic model. Others optimize communication cost when selecting a new location for replica placement [2] within a single administrative domain.

Similar to OceanStore, the Intermemory project [18] uses Cauchy Reed-Solomon Codes to archive wide scale durability. We anticipate that our combination of active and archival object forms will allow greater update performance while retaining Intermemory's survivability benefits.

7 CONCLUSION

The rise of ubiquitous computing has spawned an urgent need for persistent information. In this paper we presented OceanStore, a utility infrastructure designed to span the globe and provide secure, highly available access to persistent objects.

Several properties distinguish OceanStore from other systems: the *utility model*, the *untrusted infrastructure*, support for truly *nomadic data*, and use of *introspection* to enhance performance and maintainability. A utility model makes the notion of a global system possible, but introduces the possibility of untrustworthy servers in the system. To this end, we assume that servers may be run by adversaries and cannot be trusted with cleartext; as a result, server-side operations such as conflict-resolution must be performed di-

rectly on encrypted information. Nomadic data permits a wide range of optimizations for access to information by bringing it "close" to where it is needed, and enables rapid response to regional outages and denial-of-service attacks. These optimizations are assisted by introspection, the continuous online collection and analysis of access patterns.

OceanStore is under construction. This paper presented many of the design elements and algorithms of OceanStore; several have been implemented. Hopefully, we have convinced the reader that an infrastructure such as OceanStore *is* possible to construct; that it is desirable should be obvious.

8 ACKNOWLEDGEMENTS

We would like to thank the following people who have been instrumental in helping us to refine our thoughts about OceanStore (in alphabetical order): William Bolosky, Michael Franklin, Jim Gray, James Hamilton, Joseph Hellerstein, Anthony Joseph, Josh MacDonald, David Patterson, Satish Rao, Dawn Song, Bill Tetzlaff, Doug Tygar, Steve Weis, and Richard Wheeler.

In addition, we would like to acknowledge the enthusiastic support of our DARPA program manager, Jean Scholtz, and industrial funding from EMC and IBM.

9 REFERENCES

[1] M. Abadi. On SDSI's linked local name spaces. In *Proc. of IEEE CSFW*, 1997.

[2] S. Acharya and S. B. Zdonik. An efficient scheme for dynamic data replication. Technical Report CS-93-43, Department of Computer Science, Brown University, 1993.

[3] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless Network File Systems. In *Proc. of ACM SOSP*, Dec. 1995.

[4] B. Barak, A. Herzberg, D. Naor, and E. Shai. The proactive security toolkit and applications. In *Proc. of ACM CCS Conf.*, pages 18–27, Nov. 1999.

[5] M. Blaze. A cryptographic file system for UNIX. In *Proc. of ACM CCS Conf.*, Nov. 1993.

[6] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proc. of IEEE SRSP*, May 1996.

[7] B. Bloom. Space/time trade-offs in hash coding with allowable errors. In *Communications of the ACM*, volume 13(7), pages 422–426, July 1970.

[8] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *Proc. of Sigmetrics*, June 2000.

[9] W. Bolosky, R. Draves, R. Fitzgerald, C. Fraser, M. Jones, T. Knoblock, and R. Rashid. Operating systems directions for the next millennium. In *Proc. of HOTOS Conf.*, May 1997.

[10] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proc. of USENIX Symp. on OSDI*, 1999.

[11] S. Chaudhuri and V. Narasayya. AutoAdmin "what-if" index analysis utility. In *Proc. of ACM SIGMOD Conf.*, pages 367–378, June 1998.

[12] M. Dahlin, T. Anderson, D. Patterson, and R. Wang. Cooperative caching: Using remote client memory to improve file system performance. In *Proc. of USENIX Symp. on OSDI*, Nov. 1994.

[13] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. The Bayou architecture: Support for data sharing among mobile users. In *Proc. of IEEE Workshop on Mobile Computing Systems & Applications*, Dec. 1994.

[14] W. Edwards, E. Mynatt, K. Petersen, M. Spreitzer, D. Terry, and M. Theimer. Designing and implementing asynchronous collaborative applications with Bayou. In *Proc. of ACM Symp. on User Interface Software & Technology*, pages 119–128, 1997.

[15] C. Ellison, B. Frantz, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. RFC 2693, 1999.

[16] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary cache: A scalable wide-area Web cache sharing protocol. In *Proc. of ACM SIGCOMM Conf.*, pages 254–265, Sept. 1998.

[17] M. Franklin, M. Carey, and M. Livny. Transactional client-server cache consistency: Alternatives and performance. *ACM Transactions on Database Systems*, 22(3):315–363, Sept. 1997.

[18] A. Goldberg and P. Yianilos. Towards an archival intermemory. In *Proc. of IEEE ADL*, pages 147–156, Apr. 1998.

[19] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proc. of ACM SIGMOD Conf.*, volume 25, 2, pages 173–182, June 1996.

[20] J. Griffioen and R. Appleton. Reducing file system latency using a predictive approach. In *Proc. of USENIX Summer Technical Conf.*, June 1994.

[21] E. Hagersten, A. Landin, and S. Haridi. DDM — A Cache-only Memory Architecture. *IEEE Computer*, Sept. 1992.

[22] J. Hill, R. Szewczyk, A. Woo, D. Culler, S. Hollar, and K. Pister. System architecture directions for networked sensors. In *Proc. of ASPLOS*, Nov. 2000.

[23] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, Feb. 1988.

[24] J. Hughes, C. Feist, H. S, M. O'Keefe, and D. Corcoran. A universal access, smart-card-based secure file system. In *Proc. of the Atlanta Linux Showcase*, Oct. 1999.

[25] L. Kawell, S. Beckhardt, T. Halvorsen, R. Ozzie, and I. Greif. Replicated document management in a group communication system. In *Proc. of ACM CSCW Conf.*, Sept. 1988.

[26] J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, Feb. 1992.

[27] T. Kroeger and D. Long. Predicting file-system actions from prior events. In *Proc. of USENIX Winter Technical Conf.*, pages 319–328, Jan. 1996.

[28] G. Kuenning. The design of the seer predictive caching system. In *Proc. of IEEE Workshop on Mobile Computing Systems & Applications*, Dec. 1994.

[29] H. Kung and J. Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems*, 6(2):213–226, June 1981.

[30] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM TOPLAS*, 4(3):382–401, 1982.

[31] E. Levy and A. Silberschatz. Distributed file systems: Concepts and examples. *ACM Computing Surveys*, 22(4):321–375, Dec. 1990.

[32] M. Luby, M. Mitzenmacher, M. Shokrollahi, D. Spielman, and V. Stemann. Analysis of low density codes and improved designs using irregular graphs. In *Proc. of ACM STOC*, May 1998.

[33] L. Mackert and G. Lohman. R* optimizer validation and performance for distributed queries. In *Proc. of Intl. Conf. on VLDB*, Aug. 1986.

[34] J. Matthews, D. Roselli, A. Costello, R. Wang, and T. Anderson. Improving the performance of log-structured file systems with adaptive methods. In *Proc. of ACM SOSP*, Oct. 1997.

[35] D. Mazières, M. Kaminsky, F. Kaashoek, and E. Witchel. Separating key management from file system security. In *Proc. of ACM SOSP*, 1999.

[36] M. Nelson, B. Welch, and J. Ousterhout. Caching in the sprite network file system. *IEEE/ACM Transactions on Networking*, 6(1):134–154, Feb. 1988.

[37] NIST. FIPS 186 digital signature standard. May 1994.

[38] D. Norman. *The Invisible Computer*, pages 62–63. MIT Press, Cambridge, MA, 1999.

[39] J. Plank. A tutorial on reed-solomon coding for fault-tolerance in raid-like systems. *Software Practice and Experience*, 27(9):995–1012, Sept. 1997.

[40] C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of ACM SPAA*, pages 311–320, Newport, Rhode Island, June 1997.

[41] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal. A dynamic object replication and migration protocol for an internet hosting service. In *Proc. of IEEE ICDCS*, pages 101–113, June 1999.

[42] R. Rivest and B. Lampson. SDSI—A simple distributed security infrastructure. Manuscript, 1996.

[43] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun Network Filesystem. In *Proc. of USENIX Summer Technical Conf.*, June 1985.

[44] D. Santry, M. Feeley, N. Hutchinson, A. Veitch, R. Carton, and J. Ofir. Deciding when to forget in the Elephant file system. In *Proc. of ACM SOSP*, Dec. 1999.

[45] M. Seltzer and C. Small. Self-monitoring and self-adapting operating systems. In *Proc. of HOTOS Conf.*, pages 124–129, May 1997.

[46] J. Sidell, P. Aoki, S. Barr, A. Sah, C. Staelin, M. Stonebraker, and A. Yu. Data replication in Mariposa. In *Proc. of IEEE ICDE*, pages 485–495, Feb. 1996.

[47] D. Song, D. Wagner, and A. Perrig. Search on encrypted data. To be published in *Proc. of IEEE SRSP*, May 2000.

[48] M. Spreitzer, M. Theimer, K. Petersen, A. Demers, and D. Terry. Dealing with server corruption in weakly consistent, replicated data systems. In *Proc. of ACM/IEEE MobiCom Conf.*, pages 234–240, Sept. 1997.

[49] M. Stonebraker. The design of the Postgres storage system. In *Proc. of Intl. Conf. on VLDB*, Sept. 1987.

[50] M. Weiser. The computer for the twenty-first centry. *Scientific American*, Sept. 1991.

[51] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems*, pages 108–136, Feb. 1996.

[52] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos operating system. In *Proc. of ACM SOSP*, pages 256–269, Dec. 1993.